UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Callosa en Fiestas

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Jaume Ferrando Berenguer
Tutor: Santiago Escobar Román

Curso 2017-2018

Resum

Callosa en Festes és un projecte de renovació d'una aplicació informativa de les festes populars 'Moros i Cristians' del poble Callosa d'en Sarrià. Tracta d'actualitzar la tecnologia, afegir una part persistent amb una base de dades i renovar la interfície d'usuari. S'utilitza la ferramenta Ionic per tal d'aconseguir-la fer híbrida, tant per a iOS com per a Android. A més inclourà una característica nova que consisteix en la possibilitat d'editar la informació de tota l'aplicació. Per el qual es tracta d'una aplicació completa per a un projecte real.

Paraules clau: Ionic, Callosa d'en Sarrià, Full-Stack, Angular, aplicació híbrida

Resumen

Callosa en Festes es un proyecto de renovación de una aplicación informativa de las fiestas populares 'Moros y Cristianos' del pueblo Callosa d'en Sarrià. Trata de actualizar la tecnología, añadir una parte persistente con una base de datos y renovar la interfaz de usuario. Se utiliza la herramienta Ionic para conseguir hacerla híbrida, tanto para iOS como para Android. Además incluirá una característica nueva que consistirá en la posibilidad de editar la información de toda la aplicación. Por lo consiguiente se trata de una aplicación completa para un proyecto real.

Palabras clave: Ionic, Callosa d'en Sarrià, Full-Stack, Angular, aplicación híbrida

Abstract

Callosa en Festes is a project of renovation of an information application of the popular celebrations 'Moros i Cristians' of the town Callosa d'en Sarrià. The theme is the update the technology to add a persistent part with a Database and the renovation of the user interface. Ionic is the tool used to make it hybrid for iOS and Android. Furthermore, it will include a new characteristic for the possibility of editing the information of the whole application. In conclusion, is is a complete application for a real project.

Key words: Ionic, Callosa d'en Sarrià, Full-Stack, Angular, hybrid application

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

Las Fiestas de Moros y Cristianos son unas populares y tradicionales fiestas de Callosa d'en Sarrià, que se vanaglorian de ser declarada de interés turístico nacional. Se celebran la segunda semana de octubre, y constan de 4 días principales, sábado, domingo, lunes y martes, pero existen multitud de actividades o actos en días anteriores o posteriores. Representan tanto la conquista musulmana de la península como la reconquista de ésta a manos de los Cristianos y mediante estos actos se conmemora, como por ejemplo con una embajada mora hacia los cristianos o la tradicional 'entrà' al ritmo de una marcha mora. Cada bando tiene sus representantes, capitán moro, capitana cristiana, embajadores etcétera y se organizan en kábilas, comunmente llamadas peñas o comparsas en otros sitios, que se van turnando anualmente los cargos importantes.

Para dar visibilidad e información, tanto para los 'festers' como para los espectadores ocasionales, se ha desarrollado esta aplicación para dispositivos Android e iOS, que consiste en una aplicación informativa sobre estas, con objetivo de tener impacto en la vida real. Consiste en una parte frontend desarrollada con Ionic, backend mediante Spring y una Base de Datos remota, con una interfaz amigable para usuarios poco familiarizados y con un usuario Administrador que pueda modificar la información de la misma directamente con la app, para así actualizarla sin necesidad de escribir código.

1.1 Motivación

Uno de los factores decisivos por el cual decidí proponer este TFG fue trabajar es un proyecto real que viera la luz. Además la temática me resulta interesante ya que es sobre unas festividades, de las que participo activamente, de mi pueblo aportando mi grano de arena a la contribución de éste. Tecnológicamente me interesaba un proyecto completo, que usara Ionic, que me permitiera aprender más sobre el desarrollo híbrido de aplicaciones y que utilizase algún framework de JavaScript, ya que cuento con conocimientos de Vue.js, un framework de JavaScript, y me permitiría una aprendizaje mas rápido.

1.2 Objetivos

El principal objetivo de la aplicación es brindar información útil y bien estructurada sobre estas fiestas. Este proyecto ha sido encargado por "La Junta de Festes de Moros i Cristians de Callosa d'en Sarrià", para renovar la aplicación existente, menos completa y difícil de mantener. Se pretende que la información mostrada sea directamente actualizable desde la aplicación, sin tener que programar los datos desde el proyecto sin compilar. Se diseñará la interfaz de manera intuitiva, para que cualquier usuario pueda navegar

libremente por ella. Permitirá actualizar la información cada año acorde a los cambios de cargos y/o cambios en el programa de fiestas.

1.3 Estructura de la memoria

- Introducción: Se hace una primera aproximación del producto, se exponen los principales motivos y lo que se espera conseguir con la realización de el proyecto para obtener una imagen general que nos ayudará a entender los siguientes capítulos.
- Estado del Arte: En este capítulo se hace un estudio de los productos semejantes en el mercado y sus características. Se centra sobre todo en la versión anterior del mismo producto.
- Análisis: se analiza las características que va a tener la aplicación, mediante el análisis de requisitos del estándar IEE 830, los casos de uso de la aplicación y finalmente los mockups del producto.
- Diseño: el capítulo enfocado a cómo se va implementar la solución final y las tecnologías utilizadas, su estructura y consideraciones a tener en cuenta. Además se compara con la versión antigua.
- Ejecución de los Casos de Uso: Un ejemplo de dos secuencias de uso en un dispositivo diferente
- Conclusiones: se comenta diferentes de la realización del proyecto, se realiza con el grado y se propone cambios futuros

CAPÍTULO 2
Estado del Arte

Existen varias aplicaciones sobre las fiestas de Moros y Cristianos, evidentemente hay una como máximo por pueblo, las cuales informan sobre diferentes datos de interés sobre los festejos, como su historia, composición y horarios. Esta información está bien detallada, pero la mayoría cuentan con interfaces caóticas y diseños anticuados. Además en el mismo pueblo existe una aplicación informativa, la cual en este proyecto se va a mejorar. Pasamos ahora a analizar su versión anterior.

2.1 Callosa en Festes

Callosa en Festes en la versión anterior a este proyecto. La tecnología utilizada para esta es Ionic 1 y teniendo en cuenta que el Ionic 4 ya está en fase Beta, se podría considerar como anticuado. Además no cuenta con parte backend, por lo que todo cambio ha de ser manipulado directamente en el proyecto, por lo que cualquier cambio hecho no sería visible hasta la actualización de la misma. Su contenido es el siguiente:

- Inicio:

  Tras una pantalla de carga, en la que se muestra la publicidad patrocinadora de la aplicación se muestra una pantalla inicial con un botón a la izquierda poco visible sobre el fondo que es la portada del libro de Fiestas de ese año.

Figura 2.1: Pantalla inicial y pantalla principal

- Menú:

  Si pulsamos el botón se nos desplegará en el lado izquierdo un menú lateral en el que podremos encontrar 3 botones que corresponden a tres secciones diferentes. Justo debajo el apartado para publicidad se muestra, en pequeño, un botón de información.

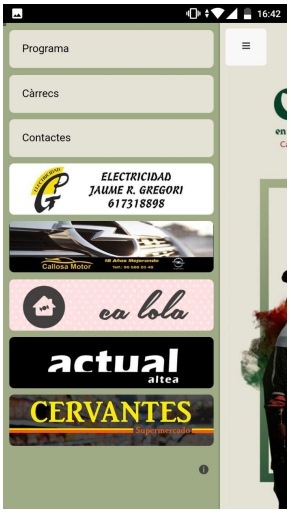

Figura 2.2: Menu

- Apartado Programa

  En este apartado se muestran los actos actuales y los próximos. Además a la izquierda arriba existe un botón para mostrar la totalidad de los actos. Si se presiona un acto se ve información detallada de éste.

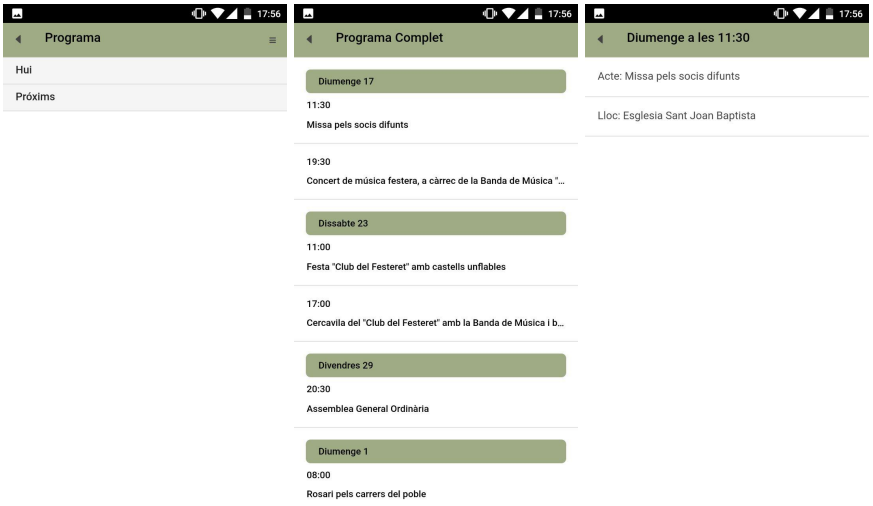

Figura 2.3: Programa

- Apartado 'Càrrecs'

  En esta página está contenida la lista de todos los cargos del año vigente, incluyendo su nombre, kábila y cargo. Si presionamos sobre uno de ellos se nos muestra su imagen agrandada. Están divididos por sus respectivos bandos: Moros i Cristianos.

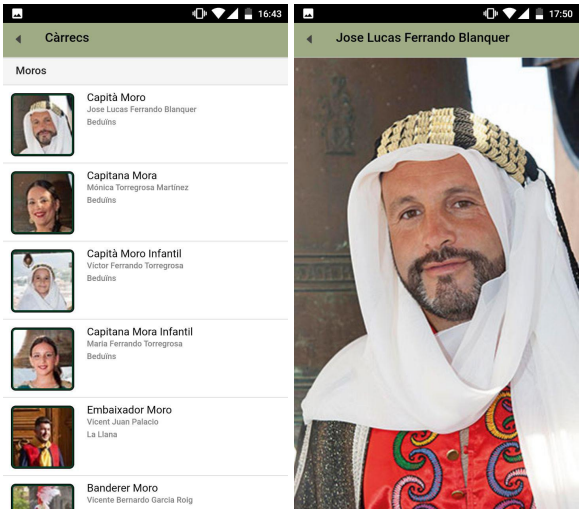

Figura 2.4: Cargos

- Apartado 'Contactes'

  Se muestra al principio los contactos de los patrocinadores, seguidos de un buscador y de un listado de todos los comercios del Pueblo.

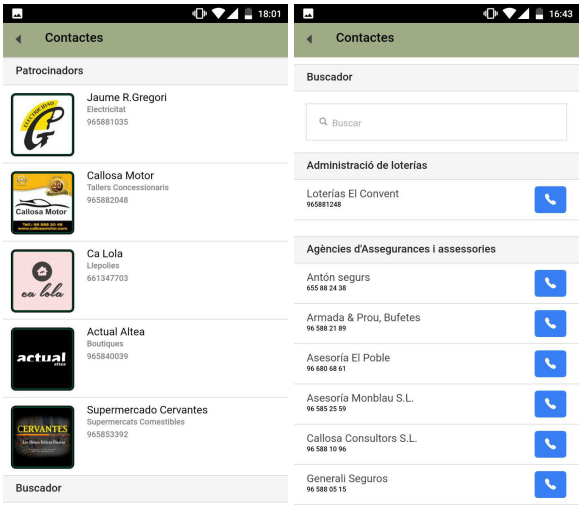

Figura 2.5: Contactos

Como podemos ver la navegación es un tanto liosa y con una estructura poco clara. Además al no contar con persistencia, como hemos dicho anteriormente, la convierte en obsoleta.

2.2 Otras aplicaciones del Mercado

- Moros i Cristians (Ontinyent)

La aplicación homóloga de Ontinyent, también informativa, nos permite conocer los diferentes cargos del año, diferentes noticias, la historia de las diferentes 'kábilas' o comparsas y el horario y sus actos, permitiendo ver dónde se va a realizar cada uno. La aplicación contiene información interesante, pero al estar mal distribuida y con un diseño muy poco profesional estos contenidos se diluyen y es poco intuitivo como acceder a ellos. La información sobre los cargos está actualizada, pero la parte de noticias parece ser que está desactualizada desde 2014. Además la página principal se podría omitir, ya que con la barra de navegación podemos acceder a la mayoría del contenido, salvo al Facebook, Twitter y a la Web.

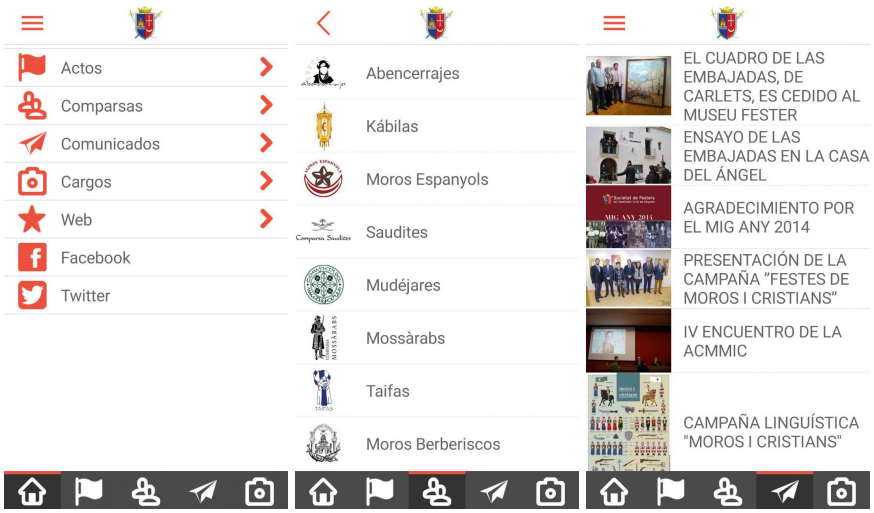

Figura 2.6: Página principal, historia y noticias

Figura 2.7: Horario y la descripción de un acto

- Moros y Cristianos (Petrer)

  En Petrer nos encontramos con esta aplicación, que al igual que la de Ontinyent dispone de mucha información útil, en este caso añadiendo información sobre comercios, sobre el mismo pueblo y en este caso toda la información está actualizada.

Sin embargo la interfaz está muy recargada, por lo que al principio cuesta trabajo identificar lo que andamos buscando.

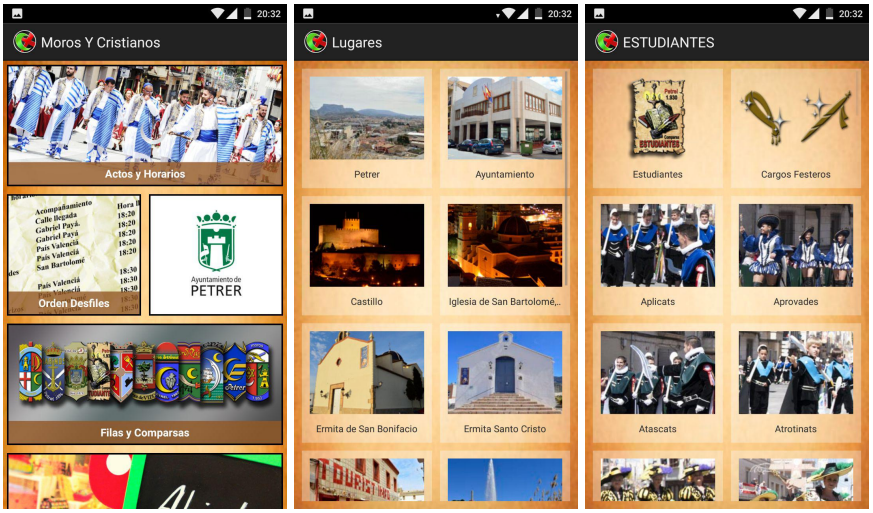

Figura 2.8: Página principal, lugares de interés y comparsa

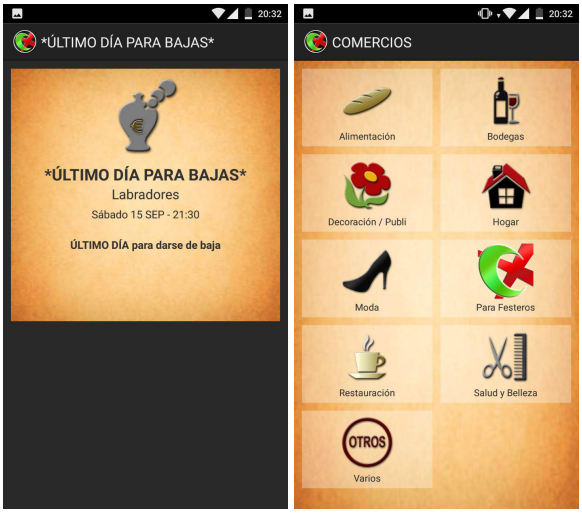

Figura 2.9: Detalle de acto y comercios

CAPÍTULO 3
Análisis

3.1 Especificación de requisitos

Según Piattini [1] *El análisis de requisitos se puede definir como el proceso del estudio de las necesidades de los usuarios para llegar a una definición de los requisitos del sistema, hardware o software, así como el proceso de estudio y refinamiento de dichos requisitos, definición proporcionada por el IEEE* y un requisito se define [1] *como una condición o capacidad que necesita el usuario para resolver un problema o conseguir un objetivo determinado*. Así, a continuación, vamos a realizar un estudio de los requisitos y analizar que condiciones habremos de satisfacer para que nuestro proyecto esté bien encaminado. Este documento ha de ser entendido por las dos partes, el Ingeniero de Software y el cliente, por lo que su lenguaje no será especialmente técnico. Este documento seguirá las pautas del estándar IEEE 830.

3.1.1. Objetivos

El objetivo de esta sección de la Memoria es redactar un documento en el que tanto el cliente como el desarrollador entiendan y estén de acuerdo, como para visualizar que características tendrá el producto para que en un futuro no haya malentendidos, ya que un cambio en desarrollo es mucho mas costoso.

3.1.2. Descripción General

En este apartado se expone el contexto de los requisitos, que afectarán tanto a ellos como al producto final. Los detalles de cada requisito se expondrán en la siguiente sección.

Perspectiva del Producto

El producto Callosa en Festes es independiente de otros productos, pero guarda estrecha relación con la pagina web de la Asociación de Moros y Cristianos y con el 'LLibre de Festes', físico. Estos productos tienen como objetivo informar sobre estas fechas, y aportar material cada año para los 'festers' con artículos y fotografías de los mismos. La función de la aplicación es ser una versión accesible y portable, que permita acceder a la información por la calle de manera funcional.

Funciones del Producto

Como hemos descrito anteriormente, la principal función del producto es informar tanto a los 'festers' como a los posibles turistas que decidan acudir al pueblo en estas Fiestas. Además permitirá mediante la interfaz de la aplicación el cambio de los datos de los cargos, horarios y demás datos de interés por un usuario Administrador. Además tendrá a la disposición de los usuarios un listado con todos los contactos de interés del pueblo.

Características de los Usuarios

Cómo usuarios solo tendremos dos tipos, ya que el objetivo de la aplicación no es que el usuario mayoritario, pueda modificar o alterar partes del proyecto. Por lo cual definiremos dos usuarios:

- Usuario no Administrador

  Este usuario es el mayoritario, ya que para acceder a la aplicación no es necesario la identificación del mismo. Puede acceder a todos los contenidos, pero no modificarlos. Incluye tanto a los 'festers' como a gente foránea del pueblo.

- Administrador

  Este usuario que ha de ser miembro de la Junta de Festes de Callosa d'en Sarrià es el encargado de actualizar, mediante la modificación de los diferentes datos, cada año mediante las herramientas que la aplicación le va a proporcionar. Este usuario ha de identificarse para entrar a su interfaz que permitirá esos cambios.

3.1.3. Requisitos Específicos

Según Raúl Monferrer[2] *Esta sección de la especificación de requisitos software contiene todos los requerimientos hasta un nivel de detalle suficiente para permitir a los diseñadores diseñar un sistema que satisfaga dichos requisitos, y que permita diseñar las pruebas que ratifiquen que el sistema cumple con las necesidades requeridas.* Como hemos aclarado anteriormente esta sección estará escrita en lenguaje natural para la fácil comprensión de cualquier lector.

Interfaces Externas

- Interfaz Usuario: La interfaz de usuario ha de ser accesible y intuitiva para cualquier tipo de usuarios, desde avanzados hasta noveles, ya que el rango de edades y profesiones es muy elevado. La navegación ha de ser clara para que sepamos en cada momento donde nos encontramos. El diseño deberá ser *responsive*, que nos permitirá que se adapte a todo tipo de dispositivos móviles, además de que según la plataforma deberá adaptarse a su estilo. Como último los colores han de ser los la Asociación de Moros y Cristianos: amarillo y negro.
- Interfaces hardware: la aplicación responderá a las pulsaciones en la pantalla y permitirá el uso del telofono mismo para llamar.

3.1.4. Requisitos Funcionales

Usuario no Administrador

Función	Consultar horario
Introducción	Visualizar el horario completo de las fiestas, con lugar y hora de cada uno
Entrada	Seleccionar en la barra de navegación inferior el icono 'horari'
Procesa	Acceso a la base de datos del orario
Salida	Muestra la página con la lista de todos los actos de fiestas

Tabla 3.1: Requisito Funcional 01

Función	Consultar información detallada de un Acto
Introducción	Visualizar los detalles de un acto de Fiestas en concreto
Entrada	Seleccionar un acto en concreto de la lista del horario
Procesa	Accede a los datos de un acto del horario
Salida	Muestra la página con los detalles de el acto en cuestión

Tabla 3.2: Requisito Funcional 02

Función	Consultar cargos
Introducción	Visualizar todos los cargos de fiestas, tanto moros como cristianos
Entrada	Seleccionar en la barra de navegación inferior el icono 'càrrecs'
Procesa	Acceso a la base de datos de los cargos
Salida	Muestra la página con la lista de todos los cargos de fiestas

Tabla 3.3: Requisito Funcional 03

Función	Consultar información detallada de un cargo
Introducción	Visualizar los detalles de un cargo de fiestas en concreto
Entrada	Seleccionar un cargo en concreto de la lista de cargos
Procesa	Accede a los datos de un cargo de la lista
Salida	Muestra la página con los detalles de el cargo en cuestión

Tabla 3.4: Requisito Funcional 04

Función	Consultar historia
Introducción	Visualizar el texto y las fotos de la historia de las Fiestas desde 1860
Entrada	Seleccionar en la barra de navegación inferior el icono 'Història'
Procesa	Accede a los datos y imágenes integrados en la aplicación
Salida	Muestra la página con la historia y diversas fotos

Tabla 3.5: Requisito Funcional 05

Función	Consultar Contactos
Introducción	Visualizar la lista con todos los contactos de interés del pueblo
Entrada	Seleccionar en la barra de navegación inferior el icono 'Contactes'
Procesa	Accede a la base de datos de los contactos
Salida	Muestra la página con una lista de los contactos del pueblo

Tabla 3.6: Requisito Funcional 06

Función	Buscar un acto
Introducción	Buscar un acto en concreto
Entrada	Seleccionar en la barra de navegación inferior el icono 'Horari' y seleccionar la barra buscadora para introducir la palabra buscada
Procesa	filtra la lista del horario
Salida	Muestra los actos que encajan con la palabra buscada

Tabla 3.7: Requisito Funcional 07

Función	Buscar un contacto
Introducción	Buscar un contacto en concreto
Entrada	Seleccionar en la barra de navegación inferior el icono 'Contactes' y seleccionar la barra buscadora para introducir la palabra buscada
Procesa	filtra la lista de los contactos
Salida	Muestra los contactos que encajan con la palabra buscada

Tabla 3.8: Requisito Funcional 08

Función	Llamar un contacto
Introducción	hacer una llamada a un contacto
Entrada	Seleccionar en la barra de navegación inferior el icono 'Contactes' y presionar un contacto
Procesa	selecciona el número del contacto
Salida	Usa el dispositivo para llamar al número marcado

Tabla 3.9: Requisito Funcional 08

Administrador

Función	Editar un acto
Introducción	Cambiar un acto en concreto
Entrada	Seleccionar en la barra de navegación inferior el icono 'Horario' y seleccionar el icono de editar
Procesa	Enviar una petición de cambio a la base de datos
Salida	El acto editado cambia la información mostrada

Tabla 3.10: Requisito Funcional 09

Función	Editar un cargo
Introducción	Cambiar un cargo en concreto
Entrada	Seleccionar en la barra de navegación inferior el icono 'Càrrecs' y seleccionar el icono de editar
Procesa	Enviar una petición de cambio a la base de datos
Salida	El cargo cambia la información mostrada

Tabla 3.11: Requisito Funcional 10

Función	Editar un contacto
Introducción	Cambiar un contacto en concreto
Entrada	Seleccionar en la barra de navegación inferior el icono 'Contactes' y seleccionar el icono de editar
Procesa	Enviar una petición de cambio a la base de datos
Salida	El contacto cambia la información mostrada

Tabla 3.12: Requisito Funcional 11

Función	Acreditarse
Introducción	Acceder a los servicios de administrador
Entrada	Seleccionar en la barra de Navegación inferior el icono 'Inici' y seleccionar Identificarse
Procesa	Envía las credenciales al backend y devuelve si puede acceder o no
Salida	Devuelve si el acceso ha resultado exitoso o no

Tabla 3.13: Requisito Funcional 12

3.2 Casos de Uso

Lso casos de uso describen las diferentes tareas que la aplicación puede hacer, y los pasos necesarios para llegar a ellas. Son un elemento muy común en esta fase en un documento de Ingeniería del Software. En este caso se han clasificado según el actor, o entidad que realiza una acción. Cada caso de uso contará de 5 apartados describiendo así sus objetivos, su función y finalmente las diferentes acciones a realizar para completarlo.

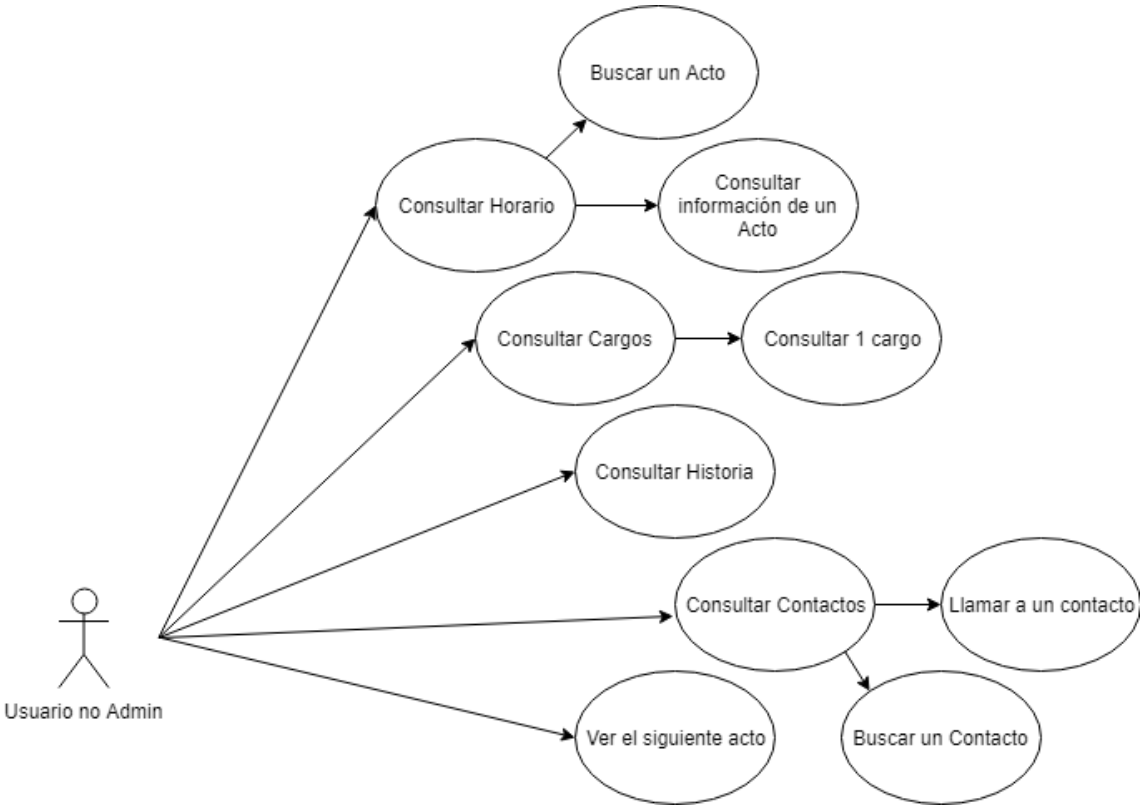

Figura 3.1: Casos de uso Usuario no Administrador

3.2.1. Usuario no Administrador

1. Código: CU 01
   Listar el programa de fiestas
   Objetivos: Consultar el horario
   Descripción: Permite visualizar todo el programa de las Fiestas
   Secuencia: Se selecciona en la barra de navegación el botón de 'Horari'. Se muestra a continuación todo el horario.

2. Código: CU 02
   Ver información de un acto
   Objetivos: Consultar la información de un acto
   Descripción: Permite la visualización de la información detallada de un acto contenido en el Programa
   Secuencia: Se selecciona en la barra de navegación el botón de 'Horari'. Se muestra a continuación todo el horario. A continuación se selecciona el acto que se desea consultar

3. Código: CU 03
   Buscar un acto
   Objetivos: Encontrar un acto
   Descripción: Permite la búsqueda de un acto por una palabra clave
   Secuencia: Se selecciona en la barra de navegación el botón de 'Horari'. Se muestra a continuación todo el horario. Seguidamente se busca en la barra de búsqueda la palabra clave.

4. Código: CU 04
   Buscar un contacto
   Objetivos: Encontrar un contacto
   Descripción: Permite la búsqueda de un contacto por una palabra clave
   Secuencia: Se selecciona en la barra de navegación el botón de 'Contactes'. Se muestra a continuación todo los contactos. Seguidamene se busca en la barra de búsqueda la palabra clave.

5. Código: CU 05
   Listar la lista de cargos
   Objetivos: Consultar los cargos de fiestas
   Descripción: Permite visualizar los cargos de fiestas de el año que se consulte
   Secuencia: Se selecciona en la barra de navegación el botón de 'Càrrecs'. Se muestra a continuación una lista con todos los cargos.

6. Código: CU 06
   Ver información de un cargo
   Objetivos: Consultar la información de un cargo
   Descripción: Permite visualizar la información detallada de un cargo de las Fiestas
   Secuencia: Se selecciona en la barra de navegación el botón de 'Càrrecs'. Se muestra a continuación una lista con todos los cargos. A continuación se selecciona el cargo que se desea consultar.

7. Código: CU 07
   Ver la historia de Fiestas
   Objetivos: Consultar la historia de las fiestas
   Descripción: Permite visualizar la historia de las fiestas de Callosa d'en Sarrià
   Secuencia: Se selecciona en la barra de navegación el botón de 'Història". Se muestra a continuación la historia.

8. Código: CU 08
   Listar los contactos
   Objetivos: Consultar los contactos
   Descripción: Permite visualizar todos los contactos relevantes del pueblo
   Secuencia: Se selecciona en la barra de Navegación el botón de 'Contactes'. Se muestra a continuación todos los contactos

9. Código: CU 09
   Llamar a un contacto
   Objetivos: Telefonear a un contacto
   Descripción: Permite llamar utilizando el marcador del móvil a un contacto listado
   Secuencia: Se selecciona en la barra de Navegación el botón de 'Contactes'. Se muestra a continuación todos los contactos y luego se presiona el botón con forma de teléfono.

10. Código: CU 010
    Ver el siguiente acto
    Objetivos: Consultar el siguiente acto
    Descripción: Permite visualizar el acto de fiestas que irá a continuación
    Secuencia: Se selecciona en la barra de Navegación el botón de 'Home'. Se muestra a continuación la pantalla principal y en un apartado se puede consultar.

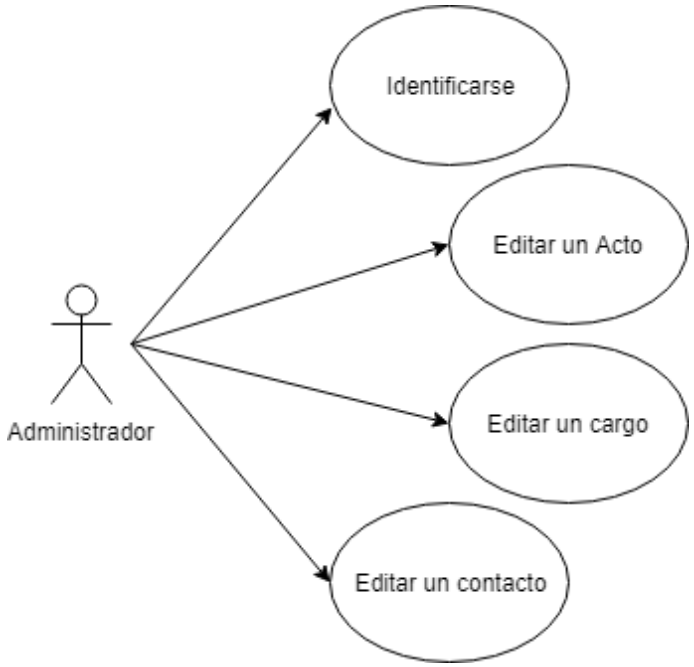

Figura 3.2: Casos de uso Usuario Administrador

3.2.2. Administrador

1. Código: CU 011
   Identificarse
   Objetivos: Acreditarse como usuario Administrador
   Descripción: Permite el acceso a las herramientas de administrador de la aplicación
   Secuencia: Se selecciona en la barra de Navegación el botón de 'Home'. Se muestra a continuación la pantalla principal y se selecciona el apartado login.

2. Código: CU 012
   Editar un acto
   Objetivos: Editar un acto en concreto
   Descripción: Permite editar un acto en la base de datos
   Secuencia: Se selecciona en la barra de Navegación el botón de 'Horari'. Se muestra a continuación la pantalla del horario y se selecciona el botón de editar.

3. Código: CU 013
   Editar un cargo
   Objetivos: Editar un cargo en concreto
   Descripción: Permite editar un cargo en la base de datos
   Secuencia: Se selecciona en la barra de Navegación el botón de 'Càrrecs'. Se muestra a continuación la pantalla de los cargos y se selecciona el botón de editar.

4. Código: CU 013
   Editar un contacto
   Objetivos: Editar un acto en contacto
   Descripción: Permite editar un contacto en la base de datos
   Secuencia: Se selecciona en la barra de Navegación el botón de 'Contactes'. Se muestra a continuación la pantalla de los contactos y se selecciona el botón de editar.

3.3 Mockups

Los mockups son el conjunto de diseños sobre los cuales se va a basar el diseño final de la aplicación. Son la idea conceptual del diseño, teniendo mas en cuenta la estructura que y la forma que el contenido. Cada mockup tiene relacionado uno o varios casos de uso, ya que a través de las pantallas que representa se pueden realizar una o varias acciones, además se diseña ya la navegación entre páginas. *Al trabajar con un mock-up, el cliente expone ante el profesional qué es lo que pasa por su cabeza, y consigue comprender mejor el nivel de complejidad del trabajo, lo cual también facilita la comunicación y desarrollo del proyecto luego de la contratación*. [3]

3.3.1. No Administrador

Inicio

Esta es la página principal de la aplicación y la primera en aparecer. Contendrá el acceso a la identificación, una pantalla de bienvenida la publicidad y el acto que está en curso.

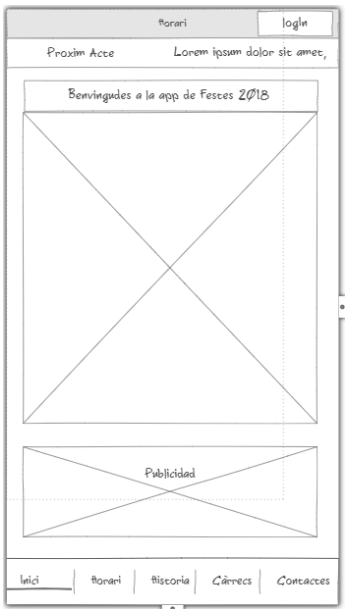

Figura 3.3: Mockups Inici, incluye el caso de uso CU 10.

Horario

Es en página se muestra una lista con el horario general con todos los actos de las fiestas. Arriba tiene un buscador, que nos permite filtrar para buscar el acto deseado, si pulsamos cualquier acto se nos redirige a la información detallada de el acto seleccionado. Esta pantalla nos muestra una imagen del acto más una descripción del mismo.

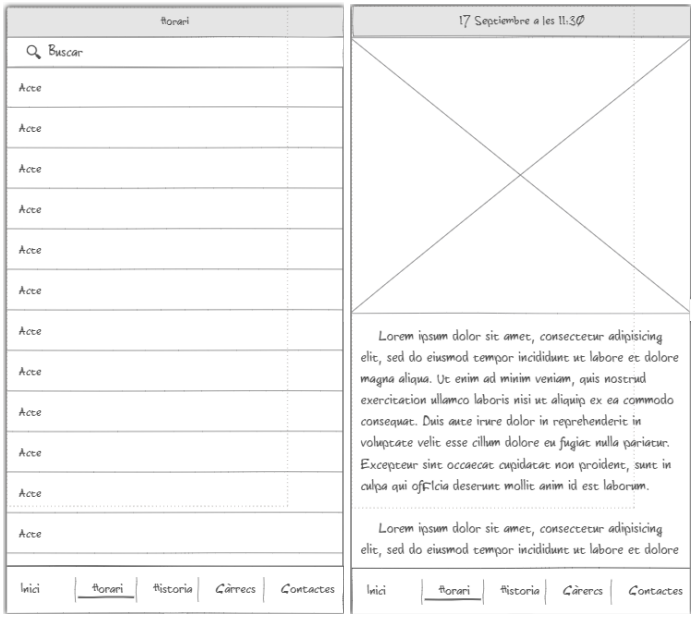

Figura 3.4: Mockups Horario, incluye los casos de uso CU 01, CU 02, CU 03.

Historia

El contenido de História es un paseo por la historia de las fiestas del pueblo, que se remontan a 1860, seguido de imágenes de época y finalmente un apartado con datos actuales.

Figura 3.5: Mockup Historia, incluye el caso de uso CU 07.

Cargos

Se muestra una lista por cada bando de las fiestas, el moro y el cristiano, con los nombres de los diferentes cargos. Si se presiona cualquiera se abre una nueva página con la información detallada con una fotografía de el mismo.

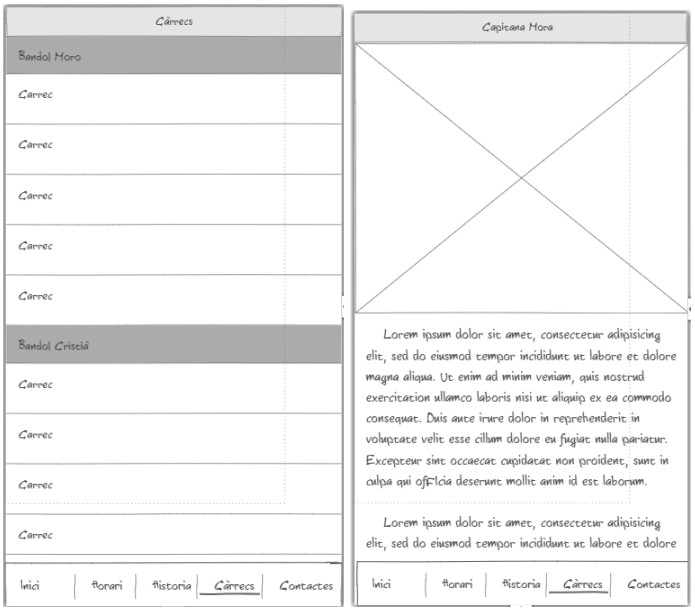

Figura 3.6: Mockups Cargos, incluye los casos de uso CU 05, CU 06.

Contactos

La página se compone de un listado de todos los contactos del pueblo, organizado por categorías. Cada contacto, al presionarlo, nos permite llamar al susodicho. Finalmente en la barra superior nos permite buscar un contacto que deseemos buscar.

Figura 3.7: Mockups Contactos, incluye los casos de uso CU 04, CU 08, CU 09.

3.3.2. Administrador

Identificarse

Esta ventana emergente de la pagina principal permite a usuario acreditarse para entrar al modo administrador

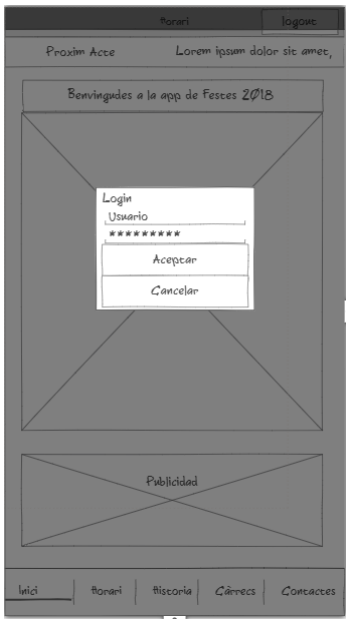

Figura 3.8: Mockup Identificarse, incluye el caso de uso CU 011.

Editar Actos

En la vista de 'horari' de Administrador se podrá editar cada acto con nuevos datos. Contará con una ventana emergente con los datos ya almacenados.

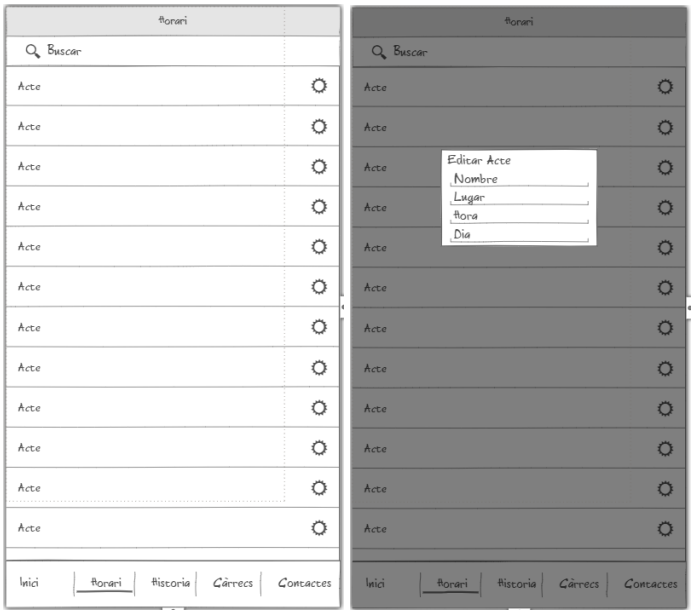

Figura 3.9: Mockups editar actos, incluye el caso de uso CU 12.

Editar editar Cargos

En la vista de 'Carrecs' de Administrador se podrá editar cada cargo con nuevos datos. Contará con una ventana emergente con los datos ya almacenados.

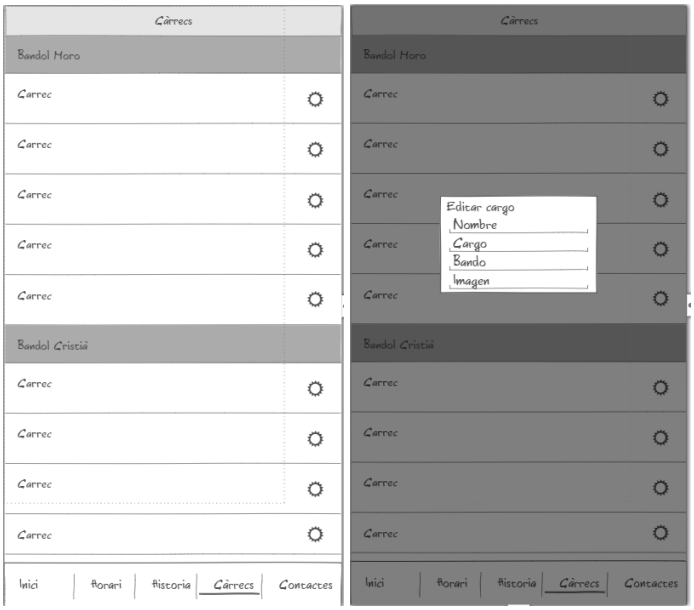

Figura 3.10: Mockups editar cargos, incluye el caso de uso CU 13.

Editar editar Contactos

En la vista de 'Contactes' de Administrador se podrá editar cada contacto con nuevos datos. Contará con una ventana emergente con los datos ya almacenados.

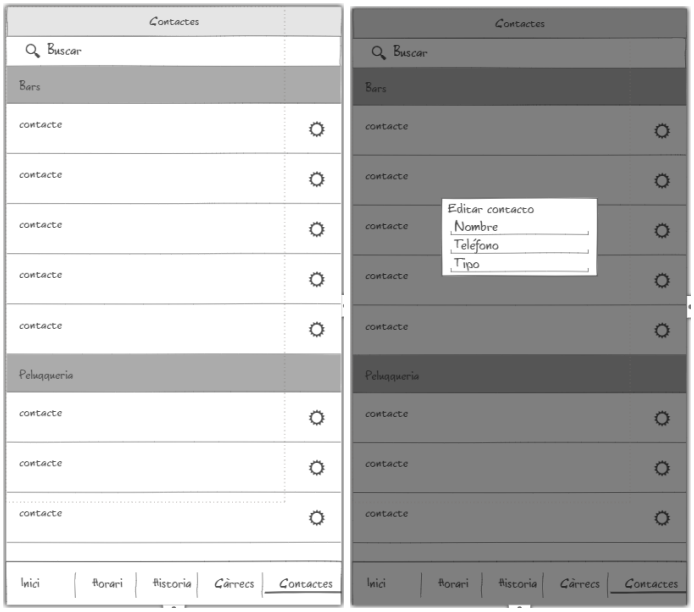

Figura 3.11: Mockups editar contactos, incluye el caso de uso CU 14.

CAPÍTULO 4
Diseño

4.1 Tecnologías

En los últimos años las aplicaciones web han experimentado un auge notable a costa de las aplicaciones de escritorio. Las aplicaciones web son herramientas software que tienen como principal característica el acceso a ellas mediante un navegador web, ya estando conectados a Internet o bien a una Intranet, y esto les supone la increíble ventaja de no ser necesaria su instalación. Este crecimiento ha permitido el desarrollo de multitud de herramientas para su creación, o como la tecnología que se usara en este proyecto, para usar sus herramientas para crear proyectos derivados. Usualmente las aplicaciones web se componen en su parte front-end de una combinación de HTML, JavaScript o un framework de este y CSS. El front-end es el software que se ejecuta y es interaccionado por el cliente, mientras que en la parte del servidor tenemos el backend, donde se almacena la lógica de la aplicación y la conexión con la Base de Datos, donde se encuentra la parte persistente de ésta.

Cuando quieras crear un producto desde cero, la mejor opción es ir a la solución más fácil.[15], que quiere decir que es preferible utilizar tecnologías que nos faciliten el trabajo, con preconfiguraciones, y tener la aplicación final lista a tiempo que tener que hacer todo el trabajo. Esto nos permite escalar fácilmente nuestra aplicación y cuando se requiera ya migrar a tecnologías. *Está bien si la tecnología para la prueba de concepto termina siendo diferente a la usada al final*[15]. Sin embargo hay que tener en cuenta qué se va a hacer y al mismo tiempo que elegimos tecnologías que nos faciliten el trabajo hay que elegirlas que se correspondan al trabajo que van a desempeñar y la complejidad del proyecto final como se comenta en este artículo *La primera cosa para decidir un tipo de aplicación web que estás desarrollando. Una pila tecnológica es un conjunto de herramientas para crear una aplicación web, necesitando una completa visión en conjunto de lo que estás planificando construir para elegir las herramientas apropiadas. Debes encontrar un conjunto de herramientas que te proporcione ventajas únicas para tu aplicación web*[16]. Un balance entre escoger la tecnología más apropiada para el tipo de proyecto deseado y una que te permita desarrollarlo rápido y tener una versión final a tiempo, y teniendo en cuenta la complejidad de el proyecto, sería la clave para optimizar tiempo y adecuarse a el producto requerido.

4.1.1. Ionic

Se podria decir que Ionic es la piedra angular de este proyecto, ya que nos permite construir de manera eficiente y con poco coste una aplicación híbrida. Ionic es un framework para el desarrollo de aplicaciones híbridas, con foco en los móviles y tabletas, aunque también puede usarse en aplicaciones web. Está basado en Angular, usando una

Figura 4.1: Ionic logo

versión diferente según la de Ionic. El cambio más significativo fue de Ionic 1 a Ionic 2, que cambió AngularJs a Angular 2, usando este último Typescript. Más adelante explicaremos estas dos tecnologías. Ionic nos permite programar nuestra aplicación como si de una web se tratase, permitiéndonos usar los casi infinitos módulos, o componentes, disponibles en la web, junto a HTML 5 y SaSS, una extensión de CSS que nos permite mas opciones, como la creación de variables CSS globales. Además de facilitar el uso a los desarrolladores web para empezar en una proyecto para móviles, nos permite mediante Apache Cordova, otra tecnología que es usada por Ionic, compilar el código a nativo Android, iOS o Windows Phone. Además Cordova nos permite el acceso a componentes del mismo dispositivo, como la cámara o el teclado.

Ionic cuenta con su propio CLI o interfaz de líneas de comandos, que nos permite desplegar el servicio en un navegador web, compilar el código o generar automáticamente diversos elementos, como una página, un servicio o un componente.*Algo que debemos destacar es que nuestras aplicaciones estarán compuestas por un árbol de componentes, por lo que serán más fácilmente escalables y sostenibles. Podremos resolver pequeños problemas, porque los componentes están estructurados modularmente. Existen componentes para multitud de cosas; implementar un botón, realizar un sistema de navegación por tabs, selectores de fechas, etc.*[10] Los componentes son bloques de código formados por HTML, CSS y a veces JavaScript.

Vamos a hacer especial mención a un componente que es fundamental para la arquitectura de Ionic, hablamos de 'NavController'. Este componente es el encargado de facilitarnos la navegación entre páginas, llamadas 'pages' en Ionic. Además nos permite tener una pila de páginas independiente entre diferentes bloques del proyecto, haciendo la navegación mas lógica y intuitiva por la interfaz.

4.1.2. Angular

Figura 4.2: Angular logo

Angular es un framework de Javascript muy popular hoy en día, que además está respaldado por Google. La versión actual es conocida como Angular 2 frente a la versión primera que es conocida como AngularJS. Ionic 3 está basado en Angular 4, pero comúnmente las versiones posteriores a Angular 2 son conocidas por Angular o Angular 2. EL modelo que usa angular es el de Modelo Vista Controlador. Nos permite manipular el HTML y comunicarnos con el backend. Las principales Ventajas de Angular son su generación de código altamente eficiente, el uso solo de CSS y HTML y la gran modularidad que se consigue con el uso de componentes. Angular 2 utiliza TypeScript en su programación, al contrario de AngularJS que utilizaba JavaScript puro. Esto lo convierte en un framework tipado y con clases, aunque su uso es opcional.

- NgModule: Una aplicación Angular está compuesta por NgModules, que dan un cotexto de compilación a los componentes y agrupa código en módulos funcionales. Un NgModules, o varios, sirve como módulo raíz para iniciar el proyecto. Los modulos pueden importar otros módulos y viceversa. Esta práctica permite organizar el código en bloques funcionales nos permite alcanzar un nivel alto de reusabilidad y el facilitar mantenimiento de aplicaciones complejas.

- Components: Cada componente define una clase, en la que es alojada la información y la lógica, es asociado con un documento HTML que le definirá la vista del mismo. Existe al menos un componente raíz por proyecto, que conecta la jerarquía de componentes con la DOM, el Modelo en Objetos para la Representación de Documentos.

- Template: Una Template está compuesta por HTML y notaciones Angular que pueden modificar el mismo HTML. Las directivas de una Template proveen de lógica y las herramientas para conectar nuestra aplicación con la DOM. Antes de montar la vista, Angular evalúa las notaciones de la plantilla para luego modificar los elementos en la DOM. Estos elementos se pueden modificar en las dos direcciones, desde el usuario o desde nuestro código.

- Services: Los servicios son usados para compartir información o lógica por los diferentes componentes, ya que ésta no está asociada a una vista específica.

- Data binding: Angular permite enlazar datos bidireccionalmente, coordiando la plantilla con las partes del componente. Esto se consigue con marcas en el documento HTML que permiten a Angular la comunicación mutua.

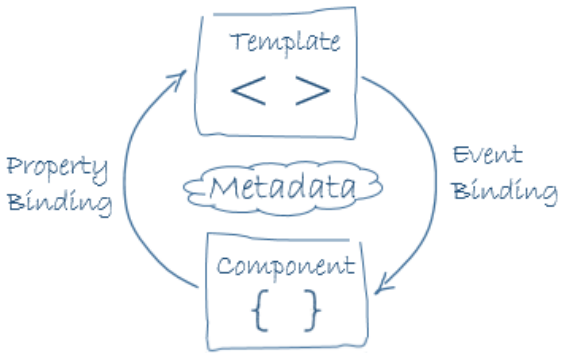

Figura 4.3: Relación entre la plantilla y el componente

4.1.3. TypeScript

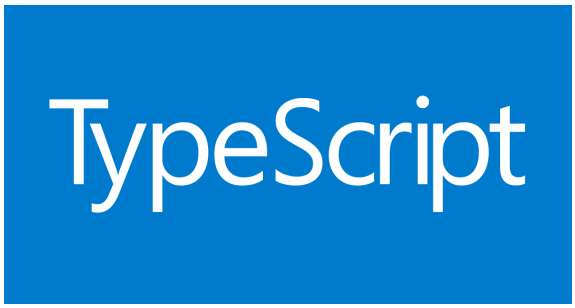

Figura 4.4: TypeScript logo

Es un lenguaje de programación orientado a objetos de código abierto desarrollado por Microsoft, que se compila en JavaScript común. Esto lo convierte en un superset, un lenguaje basado en otro lenguaje. TypeScript es especialmente adecuada para grandes proyectos, ya que es altamente escalable su código. Además es un lenguaje tipado, aunque el uso de los tipos es opcional, su uso mejora la verificación estática del código. Además ofrece el soporte de las ultimas características de JavaScript, ECMAScript 2015 y sus posibles futuras versiones, pero la compilación es hecha a ECMAScript 3, pero pudiendo seleccionar versiones mas nuevas.

4.1.4. Git

Figura 4.5: Git logo[6]

Git es un sistema de control de versiones, que es gratis y 'open source' que nos permite almacenar en un repositorio remoto las versiones de nuestro producto. Usando una series de comandos nos permite subir nuestra versión local y compararla con la versión remota y si no hya conflictos se añade en el repositorio web como una versión mueva. Mediante un sistema de versiones nos permite volver atrás y recuperar una versión antigua, además nos permite crear 'ramas' que nos permite trabajar en paralelo con otros desarrolladores o trabajar en distintas características al mismo tiempo del mismo proyecto.

4.1.5. Spring Boot

Figura 4.6: Spring Boot logo

Como todos los proyectos de la plataforma Spring IO Spring Boot utiliza como fuente la tecnología Spring. Spring es una tecnología que nos permite como desarrollador administrar la generación de objetos de distintos frameworks siendo Spring quien los va a inicializar [6], permitiendo así asegurar que se integrarán de forma adecuada. Spring Boot es una plataforma para el desarrollo rápido de una aplicación, que nos permite simplificar la creación de un proyecto Maven o Gradle y el despliegue del servidor, dejándonos centrarnos en la parte más de desarrollo, la creación de una aplicación

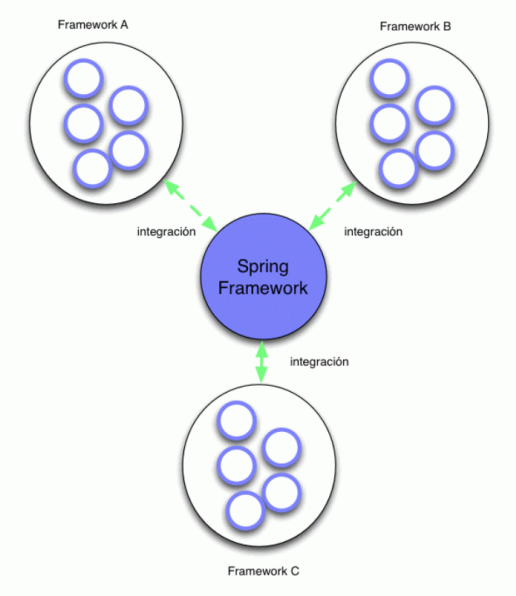

Figura 4.7: Funcionamiento de Spring

4.1.6. Hibernate

Figura 4.8: Hibernate logo

Hibernate es una herramienta de mapeo objeto-relacional para Java que nos ayuda a trazar las relaciones entre una base de datos relacional y los objetos Java, usando anotaciones en estos últimos o archivos XML. Esto quiere decir que mediante etiquetas en el proyecto Java podemos indicar que objetos son cuales en la base de datos, permitiendo una comunicación entre ambas mejor, solucionando las diferencias de los dos modelos de datos que existen en la misma aplicación. Como consecuencia aumenta la velocidad de desarrollo del proyecto.

4.1.7. MySQL

Figura 4.9: MySQL logo

MySQL es una es un gestor de base de datos *open source* muy popular entre las aplicaciones basadas en la web. Como características principales cuenta con una gran velocidad a la hora de realizar operaciones, unos requerimientos de sistema bajos, una sencilla instalación y configuración y una alta probabilidad de que los datos no se van a corromper. Ofrece otra licencia, aparte de la de *Open Source*: una Licencia comercial por Oracle Corporation, ya que está patrocinado por una empresa privada. MySQL utiliza el lenguaje SQL, por lo que MySQL gestiona bases de datos relacionales.

4.1.8. Flyway

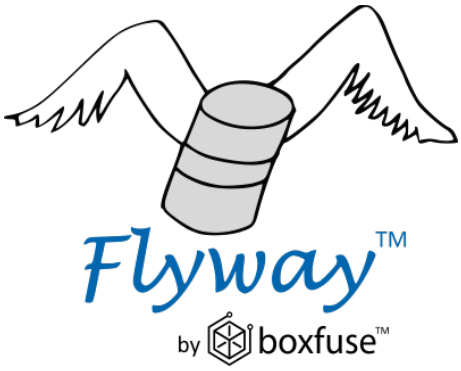

Figura 4.10: Flyway logo

Flyway es un control de versiones para las Bases de Datos SQL que nos permite migrar fácilmente entre bases de datos ya que mediante ficheros que alteran la base de datos podemos reconstruirla en otro sitio, ya que guarda el histórico desde el inicio de ésta. Las migraciones podrán ser escritas en SQL o en Java, pero en nuestro caso las escribiremos en SQL. Da soporte a las siguientes Bases de Datos[4]: Oracle, SQL Server, DB2, MySQL, PostgreSQL, CockroachDB, Redshift, Informix, H2, Hsql, Derby, SQLite, SAP HANA, Sybase ASE y Phoenix. Lo usaremos como plugin para el Spring Boot. Además es *Open Source*. Este plugin nos simplificará la tarea de migrar la base de datos a un servidor no local, o incluso a un servidor en la nube como Amazon AWS.

4.1.9. Okta

Figura 4.11: Okta logo

Okta es un servicio en la nube de administración de identidad, que nos permite administrar sistemas de identidad y simplificarlo, además de permitir escalar los productos fácilmente. Usaremos concretamente el servicio de autentificación de Okta developers. Este servicio nos permite la creación de usuarios y de grupos, la administración de estos mediante una aplicación web, la generación de tokens de autentificación, sesiones con una facilidad de uso e implementación bajas. Además está bien documentada y con tutoriales extensos. Ofrece varios planes de pago, según el número de usuarios activos al mes, incluyendo uno sin coste. Éste permite los 1000 usuarios activos al mes y teniendo en cuenta que nuestro número de usuarios registrados es uno, nos permite tener todas sus ventajas y disponer de la seguridad de sus autentificaciones sin costo alguno.

4.1.10. Docker & Kitematic

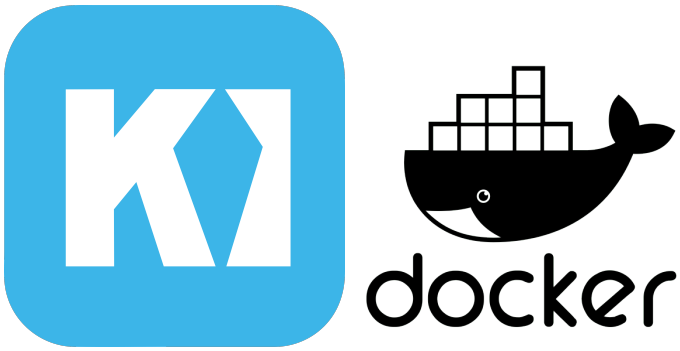

Figura 4.12: Docker & Kitematic logo

Kitematic es un proyecto *Open Source* que nos facilita la instalación de Docker y puesta en marcha automatizándola y cuenta con una interfaz gráfica para esto. Integra una *Docker Machine*, que nos permite intalar el *Docker Engine* en clietnes virtuales y su manejo mediante comandos, para proveer una Máquina Virtual *VirtualBox VM* y instalar el *Docker Engine* localmente. Kitematic nos ofrece gran cantidad de imágenes que podemos hacer correr en el momento, en nuestro caso ejecutaremos la imagen de MySQL versión 5.7 en nuestra máquina virtual. Nos proporcionará un servidor local donde alojaremos nuestra Base de Datos, que se comunicará con la aplicación creada con Spring Boot.

4.2 Entorno de Desarrollo

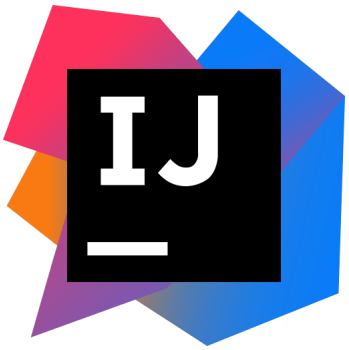

Figura 4.13: IntelliJ logo

Para realizar el proyecto se ha utilizado el entorno de desarrollo IntelliJ IDEA Ultimate Edition. IntelliJ está desarrollado por JetBrains. Su primera versión fue lanzada en 2001 siendo de los primeros entornos de desarrollo para Java que permitía refactorización y una sofisticada navegación por el código. Años mas tarde, en 2014, Google basó su entorno de desarrollo para Android, Android Studio, en este IDE. La Versión Ultimate soporta estos lenguajes: Java, Groovy, XML/XSL, Kotlin, ActionScript/MXML, CoffeeScript, HTML/XHTML/CSS, JavaScript, Ruby/JRuby y SQL y mediante plugins Clojure, Dart, Erlang, Go, Haxe, Perl, Scala, Haskell, Lua, PHP, Python y TypeScript. Soporta además estas tecnologías y frameworks: Ajax, Android, Django, EJB, FreeMarker, Google App, Engine, Google Web Toolkit, Grail, Hibernate/JPA, Java ME MIDP/CLDC, JBoss Seam, JSF, JSP, OSGi, Play, Ruby on Rails, Spring, Struts 2, Struts, Tapestry, Velocity y Web services y los siguietnes controles de revisión: CVS, Git, GitHub, Mercurial, Subversion, Team Foundation Server, ClearCase, Perforce, Visual SourceSafe, fuente [12]. Como

soporta tanto Java, Javascript, Typescript, Spring, Ajax y Git se consideró el uso de éste, además de contar con la licencia de estudiante que da acceso a la versión Ultimate. Según Sergio Delgado[13]: *Idea analiza todo tu código y mantiene su estructura permanentemente en memoria mientras estás trabajando con él. Esto tan simple, y a la vez tan potente, es la base sobre la que se construye todo el entorno. Idea sabe lo que está haciendo tu software y, probablemente, lo sabe mejor que tú.*

4.3 Estructura de la Aplicación

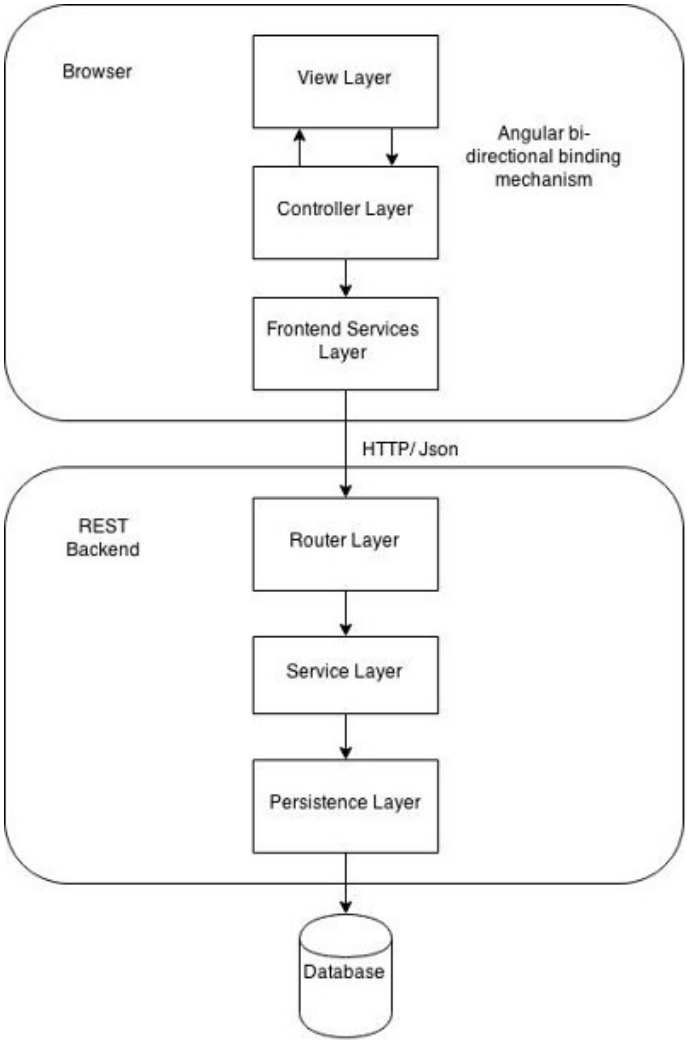

Figura 4.14: Estructura de una aplicación web[7]

La estructura de una aplicación es *el plan que describe los aspectos y decisiones que son importantes para un software. Esto implica tener en consideración todo tipo de requerimientos(rendimiento, seguridad, etc.), la organización del sistema, como las partes del sistema se comunican entre ellas, si hay algunas dependencias externas, cuales pautas y implementaciones de tecnología hay, que riesgos hay que tener en consideración y mucho más*[9] Dividiremos la estructura en diferentes capas, frontend, backend y Basde de Datos. En esta arquitectura, frente a las más tradicionales, se saca el controlador y se pasa a la parte cliente, dejando el servidor sólo para hacer peticiones REST sin estado. La capa cliente con estructura MVC, o Modelo Vista Controlador contiene toda la lógica de la presentación. La comunicación entre cliente y servidor se consigue mediante objetos JSON. La estructura MVC separa

los datos y la lógica de su presentación, por lo que encontramos en el frontend una semiestructura que consta de tres partes.El modelo es el *Frontend Services Layer* o Capa del frontend de servicios, que comunica con el servidor. Se define como la parte encargada de los datos y su gestión. La Capa Controladora o *Controller Layer* es la parte que mantiene unidas a la Vista y al Modelo. Inicializa la vista modelo y define como la vista ha de reaccionar a los cambios del Modelo. Por útimo tenemos la *View Layer* o Capa de Vistas que representa los diferentes componentes de la interfaz de usuario y de la forma en que los datos se muestran.
A continuación vamos a explicar las diferentes capas de la arquitectura de este proyecto y cómo se organizan sus diferentes componentes además de su explicación.

4.3.1. Frontend

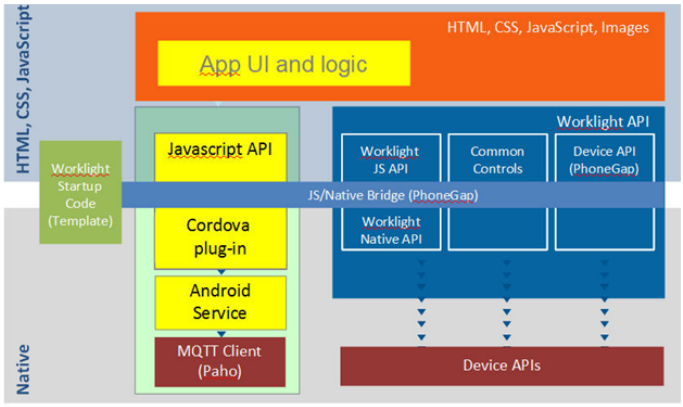

Figura 4.15: Estructura del frontend de una aplicación híbrida[5]

Es la parte del código que se ejecuta en el lado del cliente, normalmente usando HTML, CSS y Javascript y se ejecuta en el navegador y crea la interfaz de usuario. Pero en este proyecto es un poco diferente. Si que vamos a programar usando estas tres herramientas, más bien usando frameworks, pero en nuestro producto final estará en código de Android/IOs nativo, ya que mediante Ionic Cordova nos traducirá todo ese código. Sin embargo la estructura que seguirá este bloque será parecida a la usada por Angular, en este caso la que propone Ionic.

- Page: Cada vista tendrá asociada una pagina, o Page como la llama Ionic, que contará de 4 ficheros: .htlm, lmodule.ts, .scss y .ts. El fichero html contiene todo el esqueleto de la presentación de la página y será decorado con el fichero scss. La lógica es llevada por el fichero .ts que se encargará de proporcionar al html mediante el 'data-binding' los datos que necesita y comunicarse con otros componentes. Una página puede integrar en si varios componentes y/o servicios que serán declarados en el fichero module.ts.

  En el caso de la página contactos tenemos un fichero html llamado contactes.html con una lista, que se pintará usando los datos proporcionados por el fichero contactes.ts mediante los *data bindings*. Estos datos se solicitarán al iniciarse la vista, ya que no están en la aplicación, a un *provider* que se encargará de obtenerlos. Tras obtener todos los datos necesarios, contactes.html hará servir el fichero contactes.scss para decorarse. Finalmente si se presiona el icono de teléfono se llamará usando una biblioteca nativa de Ionic a un número correspondiente al elemento de la lista obtenida anteriormente. (Véase Apéndice Elementos del frontend, sección Page)

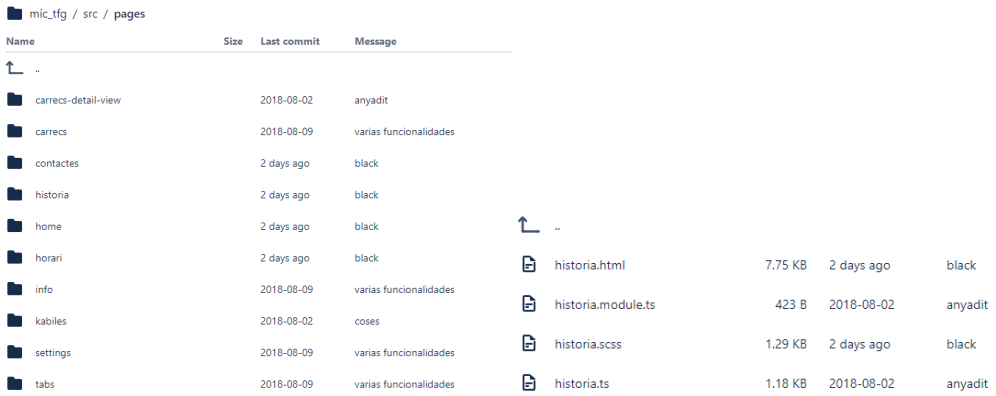

Figura 4.16: Páginas

- Directives: el rol de una directiva es alterar el comportamiento de un componente, básicamente una directiva es un componente sin una hoja HTML que nos permite modificar el componente, o página. Mediante el fichero directives.module.ts exportamos todas las directivas necesarias.

  Por ejemplo hemos usado una directiva llamada *Parallax-Header* que nos permite hacer un efecto *parallax* a la cabecera de una página.

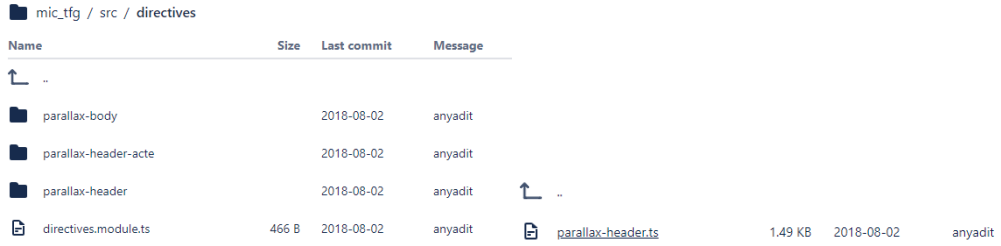

Figura 4.17: Directivas

- Pipes: la función de una Pipe es la manipulación de la información antes de ser mostrada en pantalla, permitiendo así filtrar contenidos. Como las directivas tenemos el pipes.module.ts que nos permite exportar las pipes.

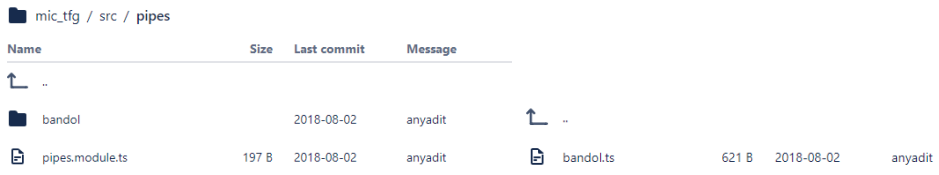

Figura 4.18: Pipes

- Services, o Providers para Ionic: los servicios son las clases que se usarán para conectar la aplicación angular con el backend, y serán llamadas por las diferentes páginas para acceder a estos datos. Son archivos .ts. Utiliza el módulo HttpClient y Observables para realizar las diferenes peticiones y devuelve el observable, que posteriormente procesará la page que los solicite.

  Siguiendo el ejemplo de las páginas, cuando el fichero contacts-data.ts ejecuta el método de obtener los contactos, se envía una petición asíncrona *get* mediante el HttpClient a la ruta donde recibe la parte del backend, y al recibirla la devuelve a la página. (Véase Apéndice Elementos del Frontend, sección Provider)

Figura 4.19: Providers Actualiar

- Archivos importantes
  - constants.ts: este provider provee de las constantes de toda la aplicación para que estén almacenadas en un mismo archivo y se nos facilite el cambio de una de estas. Por ejemplo para cambiar la ruta en la cual accede al backend de local a un servidor externo es más eficiente si la tenemos en un solo sitio guardada.
  - carpeta tabs: este componente es el que se encarga de facilitarnos la navegación a través de la aplicación y es el primero en cargarse, además de definir las diferentes pilas de navegación. Siempre va a estar presente, estando en cualquier pantalla dentro de la app.
  - variables.scss: en este archivo scss están almacenadas todas las variable globales de css, sobre todo de colores. Aqui están los colores principales, secundarios y de alerta de toda la app. Además sirve para sobreescribir las constantes de Ionic globales para personalizar más nuestra aplicación, incluyendo las variables exclusivas de cada diseño: ios, md(Material design utilizado por Android) y Windows Phone.
  - app.module.ts: Aquí se declaran todos los componentes y servicios utilizados a través de toda la aplicación y los componentes de entrada. Además cuenta con todas las importaciones.
  - carpeta platforms: tiene dos subcarpetas, android e ios. Aqui se almacenan los dos proyectos compilados Android e iOS(o incluso Windows phone) en su código fuente compilados por Cordova.
  - package.json: se almacenan todas las dependecias de la aplicación y sus versiones, tanto de angular, de Cordova y externas.

4.3.2. Backend

Este bloque de código se ejecuta en la parte del servidor, recibiendo peticiones del cliente y mediante su lógica procesarla, además se comunica con la Base de Datos donde toda la parte persistente de la aplicación esta guardada. El código de éste está en un proyecto diferente al del frontend, ya que está pensado para ejecutarse en un servidor remoto. Ha sido programado en Java, usando Spring Boot, explicado anteriormente. Cuenta con la siguiente estructura:

- Carpeta rest: Esta carpeta contiene todos los controladores, que reciben las peticiones del frontend y ejecuta los métodos de los DAO para realizar las tareas pertinentes. En el fichero están definidos los diferentes *enpoints*, normalmente con los tres básicos. Un *get all*, que obtiene todos los elementos de una tabla, un *post* que nos permite añadir un dato a la Base de Datos y un *get by id*, mediante el cual con una id de un objeto lo devolvemos de la Base de Datos.

  Siguiendo la traza en la que hemos hecho una elipsis en el apartado anterior *providers* del frontend, se recibe la petición en el archivo ContacteController.java *get* para obtener la lista entera de los contactos. Tras esto ejecuta un método acorde

de la clase ContaceDAO.java y al recibir la respuesta la devuelve.(Véase Apéndice Elementos del backend, sección Controller)

- Carpeta CAD: esta carpeta contiene las carpetas daos y intefaces. En la carpeta interfaces están contenidas las interfaces, que són una colección de métodos abstractos y de propiedades de un DAO(Data Acess Object), que contiene los métodos que ejecutará la clase Controller para realizar las operaciones CRUD. Sirve de puente entre el mismo Controller y la base de datos. Los DAO estás alojados en la carpeta daos.

  Continuado el hilo, ContaceDAO.java ejecuta el método que accede a la base de datos y recibe un la repuesta, convirtiéndola en una lista de Objetos Contacte, definido en la carpeta Model y devolviéndola al controlador. (Véase Apéndice Elementos del backend, sección DAO y Interface)

- Carpeta Model: un modelo es la representación de forma de objeto de una fila de una tabla de la Base de Datos. Contiene todos sus atributos, que serán las columnas de la tabla, y diferentes métodos que los podremos usar para procesar estos datos. Este objeto tiene asociado un tabla mediante las notacion Hibernate @TABLE y cada uno de sus atributos con una columna mediante @COLUMN, aunque es capaz de interpretarlo él sólo si coinciden los nombre incluso estando en *Camel Case*. (Véase Apéndice Elementos del backend, sección Model)

- Carpeta resources: contiene el application.properties que contiene la configuración de Spring, Hibernate y de Flyway.

- Migration (db/Migration): esta carpeta es la que usa Flyway para los ficheros SQL de su control de versiones. Cada fichero contiene una modificación de la base de datos. Estos fichero tienen una notación especial por convenio. Primero se pone la versión V1, V2.. seguido de dos barras bajas y finalmente el nombre. Así Flyway es capaz de reconocerlo. (Véase Apéndice Elementos del backend, sección Flyway Version)

4.4 Estructura de la Base de Datos

La Base de Datos consta de varias tablas en las que se clasifica y se almacena la información persistente de todo el proyecto. Para poblarla se transformó unas tablas excel con los detalles de cada elemento en JSON para posteriormente insertarla en ella. Consta del siguiente esquema:

4.4.1. Tabla actes

Es la tabla donde se almacena toda la información relacionada con el programa de fiestas. Cada fila representa un acto y consta de las siguientes columnas:

- id_acte: se almacena la id única del acto, es tipo int (10) que mediante Hibernate se transformará a int y puede no ser nulo. Es la *Primary Key*.

- nom: se almacena el nombre del acto, es tipo varchar (400) que mediante Hibernate se transformará a String y puede ser nulo. El valor por defecto es null.

- lloc: se almacena el lugar del acto, es tipo varchar (2600) que mediante Hibernate se transformará a String y puede ser nulo. El valor por defecto es null.

- hora: se almacena la hora del acto, es tipo varchar (100) que mediante Hibernate se transformará a String y puede ser nulo. El valor por defecto es null.
- mes: se almacena el mes del acto, es tipo varchar (40) que mediante Hibernate se transformará a String y puede ser nulo. El valor por defecto es null.
- dia_semana: se almacena el dia de la semana del acto, es tipo varchar (40) que mediante Hibernate se transformará a String y puede ser nulo. El valor por defecto es null.
- descripcio: se almacena la descripción del acto, es tipo varchar (5200) que mediante Hibernate se transformará a String y puede ser nulo. El valor por defecto es null.
- imatge: se almacena la ruta de la imagen del acto, es tipo varchar (400) que mediante Hibernate se transformará a String y puede ser nulo. El valor por defecto es null.
- header: se almacena si el acto es el primero del dia, es tipo bit (1) que mediante Hibernate se transformará a boolean. El valor por defecto es 0.
- posicion: se almacena la posición del acto respecto de la tabla, es tipo int (20) que mediante Hibernate se transformará a int y puede ser nulo. El valor por defecto es null.

4.4.2. Tabla carrecs

Es la tabla donde se almacena toda la información relacionada con los cargos de las fiestas. Cada fila representa un cargo y consta de las siguientes columnas:

- id_carrec: se almacena la id única del cargo, es tipo int (10) que mediante Hibernate se transformará a int y puede no ser nulo. Es la *Primary Key*.
- nom: se almacena el nombre del cargo, es tipo varchar (400) que mediante Hibernate se transformará a String y puede ser nulo. El valor por defecto es null.
- descripcio: se almacena la descripción del acto, es tipo varchar (2600) que mediante Hibernate se transformará a String y puede ser nulo. El valor por defecto es null.
- imatge: se almacena la ruta de la imagen del acto, es tipo varchar (400) que mediante Hibernate se transformará a String y puede ser nulo. El valor por defecto es null.
- carrec: se almacena la tipo de cargo, es tipo varchar (40) que mediante Hibernate se transformará a String y puede ser nulo. El valor por defecto es null.
- side: se almacena el bando del cargo, es tipo varchar (20) que mediante Hibernate se transformará a String y puede ser nulo. El valor por defecto es null.
- posicion: se almacena la posición del cargo respecto de la tabla local del bando, es tipo int (20) que mediante Hibernate se transformará a int y puede ser nulo. El valor por defecto es null.

4.4.3. Tabla contactes

Es la tabla donde se almacena toda la información relacionada con los cargos de las fiestas. Cada fila representa un contacto y consta de las siguientes columnas:

- id_contactes: se almacena la id única del contacto, es tipo int (20) que mediante Hibernate se transformará a long y puede no ser nulo. Es la *Primary Key*.

- nom: se almacena el nombre del contacto, es tipo varchar (400) que mediante Hibernate se transformará a String y puede ser nulo. El valor por defecto es null.

- descripcio: se almacena la descripción del acto, es tipo varchar (2600) que mediante Hibernate se transformará a String y puede ser nulo. El valor por defecto es null.

- telefono: se almacena el teléfono del contacto, es tipo varchar (20) que mediante Hibernate se transformará a String y puede ser nulo. El valor por defecto es null.

- tipo: se almacena el tipo de establecimiento del de contacto, es tipo varchar (800) que mediante Hibernate se transformará a String y puede ser nulo. El valor por defecto es null.

4.4.4. Tabla schema_version

Es la tabla donde se almacena toda la información relacionada las versiones del Flyway. Cada fila representa una versión de Flyway y detecta si ya han sido aplicadas o incluso si ha habido un error al aplicar una. Consta de las siguientes columnas:

- version: se almacena el número de la versión aplicada, es tipo varchar (50) que mediante Hibernate se transformará a String y puede ser nulo.

- nom: se almacena el nombre del contacto, es tipo varchar (400) y puede ser nulo. El valor por defecto es null.

- descripcio: se almacena la descripción de la versión, es tipo varchar (200) y puede no ser nulo.

- type: se almacena el lenguaje utilizado para la versión, es tipo varchar (20) y puede no ser nulo.

- script: se almacena el nombre de el fichero proporcionado por Spring, es tipo varchar (1000) y no puede ser nulo.

- checksum: permite indicar si el fichero de la versión se ha cambiado cuando se ejecuta el servidor, es tipo int (11) y puede ser nulo. El valor por defecto es null.

- script: se almacena el nombre del usuario que ha aplicado la versión, es tipo varchar (100) y no puede ser nulo.

- installed_on: se almacena la fehca de aplicación de la versión, es tipo timestamp y no puede ser nulo. El valor por defecto es CURRENT_TIMESTAMP

- execution_time: se almacena el timepo de ejecución de aplicar la versión, es tipo int (11) y no puede ser nulo.

- success: se almacena si se ejecuto la versión con éxito, es tipo tinyint (1) y no puede ser nulo.

4.5 Seguridad

Para la autentificación utilizaremos el servicio Okta, anteriormente expuesto. Mediante su servicio de Okta Developer accedemos a nuestro dominio propio autogenerado de gestión de identidades. Entre otras cosas nos ofrece estadísticas de usuarios actuales, autentificaciones y más, pero en nuestro caso poco uso le daremos ya que básicamente usaremos una pequeña fracción de la plataforma. Hemos creado un grupo Administradores donde se definirán los administradores, que por ahora constará de dos miembros: el desarrollador y el miembro de la junta que haga de gestor de la aplicación. No hemos permitido la creación de nuevos usuarios ya que sólo se precisa de administradores en la aplicación, ya que no se ofrece un servicio personalizado por usuarios, sino una ventana de edición de datos.

Desde la página principal de la aplicación accederemos al Login, que tras introducir nuestras credenciales las validará mediante los métodos de Auth0, una autentificación como servicio, que nos creará una sesión en el dispositivo por la cual accederemos alas características propias del Administrador. Esta sesión puede expirar y funciona a través de tokens, una metodología en al cual el servidor externo nos da una 'ficha' por la cual nos identifica como usuario autenticado y evita los posibles fallos de seguridad para acceder a rutas sin las credenciales.

El servicio se instala a través de npm a Ionic, i con una serie de métodos como login y conexiones al servidor de okta se configura y se accede. Implementa un servicio llamado outhService que gestionará internamente la lógica y que además nos permite usar el método outhService.hasValidToken para comprobar si estamos autenticados. Por ejemplo, en el fichero 'carrecs.html' existe el botón editar un cargo, que sólo se puede acceder si estás autenticado, y se consigue mediante este método contenido en una clausa *ngIf de Angular que nos muestra/oculta el elemento de la Dom dada una condición.

4.6 Comparativa

Como se ha explicado antes este proyecto es una versión actualizada de una versión anterior. En este capítulo vamos a exponer los cambios hechos y su motivo además de las diferentes dificultades que ha supuesto. La aplicación se ha reescrito entera de nuevo ya que se ha actualizado la tecnología y se han añadido nuevas características. Además el diseño se ha renovado enteramente conforme a la línea que siguen las aplicaciones actuales.

4.6.1. Parte tecnológica

- Parte Cliente: Aunque las dos versiones están basadas en Ionic y es fácil pensar que cambiando un poco el código se podría obtener la siguiente versión, no es así. El principal cambio entre las versiones de Ionic 1 y 3 es el cambio implícito de usar Angular 4 (o Angular) frente a AngularJS. No sólo la estructura es diferente, sino también el lenguaje. Ahora Angular 2 usa TypeScript frente a JavaScript. Como explican en campusMVP[8]: *El mal llamado Angular 2, que se llama solamente Angular, no es una nueva versión de AngularJS (también denominado Angular 1.x). Es un nuevo framework, escrito desde cero y con conceptos y formas de trabajar completamente distintos.* Además todas las notaciones cambian, desaparecen los controladores scope y el routing entre otras cosas. Como consecuencia Angular es más rapido pero su curva de apren-

dizaje aumenta. Por suerte todos estos cambios nos aportan muchas ventajas a la hora de programar, como por ejemplo el routing de Ionic 3 mediante Pages, usando push y sus pilas de navegación independientes nos facilitan programar la navegación.

AngularJS[8]

```
var myApp = angular
    .module("miModulo", [])
    .controller("productoController", function($scope, $http) {
         var prods = { name: "Queso", quantity: 5 };
         $scope.productos = prods;
     });
```

```
<input ng-model="prod.name"></input>
```

Angular 2[8]

```
import { Injectable } from '@angular/core';
import { Http } from '@angular/http';
import { Producto } from 'app/shared/Producto.ts';

@Injectable()

export class ProductosService {
    constructor(private _http: Http) { }
    getProductos() {
        return [new Producto(1, 'Queso'),
            new Producto(2, 'Pan',
            new Producto(3, 'Verdura')
        ];
    }
}
```

```
<input [(ngModel)]="prod.name"></input>
```

En cuanto a estructura del proyecto también sufre cambios, principalmente por cambia de AngualJS a Angular. La carpeta www, dónde se almacenaba toda la lógica de la aplicación ahora se traslada a la carpeta src. Al ser el lenguaje de Angular TypeScript, necesita una carpeta destino donde compilarse, que será la nueva www. Además los antiguos controllers, los ficheros html de las views y los ficheros css de la carpeta css se transladan a una carpeta conjunta, pages y se organizan entre ellos en páginas individuales, con un archivo de cada por page. Las factories serán los nuevos providers que se encargarán de proporcionar los datos a las páginas.(Figura 4.20)

- Parte Servidor: en la versión actual se incluye un servidor, mientras que en la anterior no existía, por lo que todo el backend, la Base de Datos y la comunicación entre sus capas, frontend, backend y base de datos es de nueva implementación. Anteriormente toda la infomración y imágenes estaban almacenadas dentro de la misma aplicación y para editarlas se precisaba de una actualización de Google Play o App Store, además de depender de la disponibilidad de el programadaor en cuestión. Ahora el usuario Administrador puede editar los datos sin necesidad del programador para estas tareas. Para mas detalles de la implantación ver la sección de implementación.

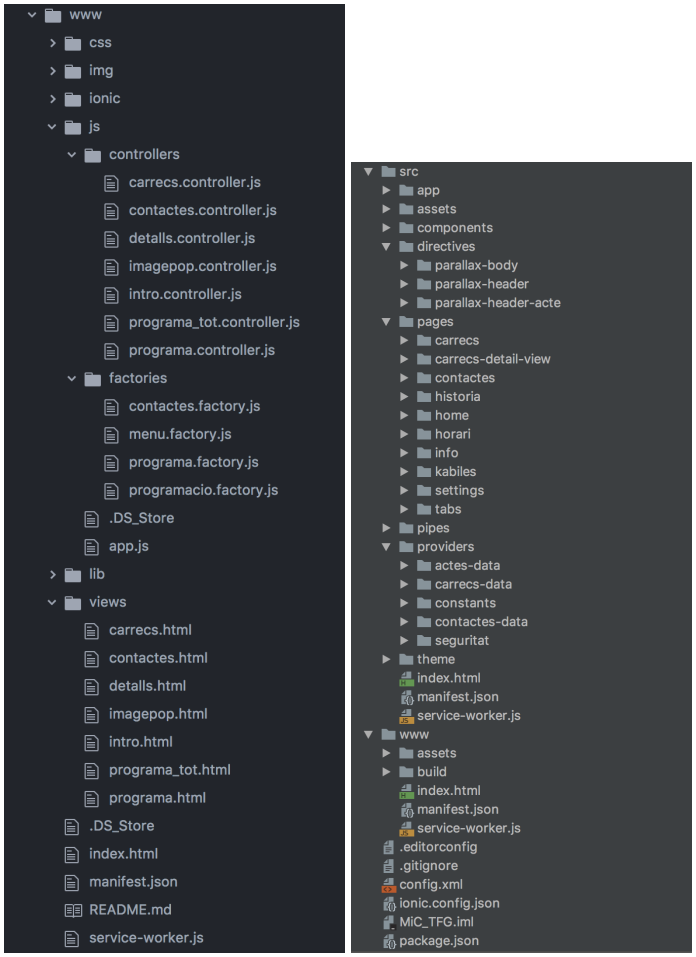

Figura 4.20: Comparativa de las dos estructuras,la primera de la versión anterior y la segunda de la nueva

4.6.2. Diseño

Según Georgia Arminsen el diseño[17] *junto con la experiencia de navegación de la app, es lo que el usuario percibe y, por tanto, lo que hay que mimar, ya que el usuario no ve todo lo que pasa por debajo del contenido. Define al diseño mobile como el puente entre el interior y el exterior y asegura que es éste el que ayuda a que la experiencia de usuario sea completa* . Por lo tanto se consideró que por diversas razones la versión anterior necesitaba un lavado de cara, ya que en cuestiones de diseño no era muy avanzada. Hemos clasificado estos cambios en tres secciones, color, navegación y diseño de página.

1. Colores: los colores de la aplicación eran el verde pistacho principal: (9fab85), su versión clara (e4e2d5), y color de fondo blanco (ffffff) se correspondían a los colores de la portada del libro de fiestas de ese año, por lo que cada año que cambiara la portada estos colores deberían cambiar lo convertía en algo poco práctico(Figura 4.20 primera imagen). Por lo tanto se decidió cambiar los colores por los de la Asociación de Moros y Cristianos: Amarillo y negro, por lo que la nueva paleta de colores se rehízo conforme estos usando los patrones de Material Design[11] y usando su herramienta para elegir las variantes de los colores de la paleta. Ahora el color principal es el Amarillo (fdd835) y el secundario el negro(212121).(Figura 6.2 segunda y tercera imagen)

   Para los fondos usaremos una versión mas clara de el color secundario, mientras que para las cabeceras y barras de navegación se utilizará el negro secun-

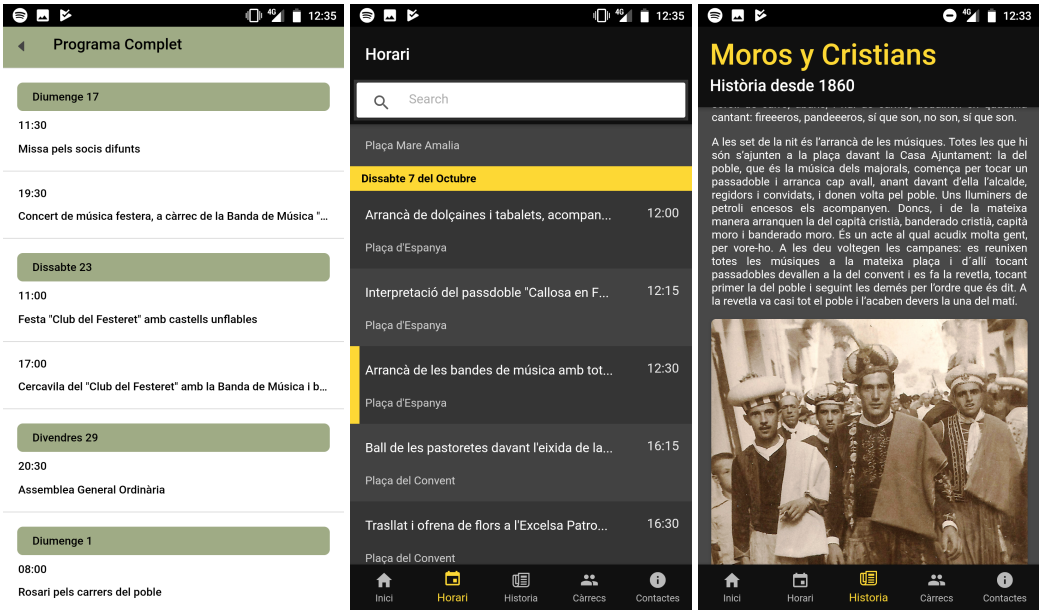

Figura 4.21: Diferencias entre las paletas de las dos aplicaciones, el primero corresponde a la versión anterior

dario. Esto implica una complicación de diseño, ya que cuadrar los colores en fondos negros suele ser mas complicado. Finalmente para el texto se optó por el blanco para mayor legibilidad y para el texto menos importante o iconos no seleccionados un blanco con un valor de alpha más bajo, o opacidad menor. El resultado es una paleta mas elegante y que nos permite actualizar la aplicación sin tener que cambiar la paleta de año en año.

2. Navegación: Para cambiar la navegación con menú lateral se ha decidido trasladar el menú a una barra de navegación, ya que de manera más intuitiva nos permite navegar por las diferentes secciones(Figura 4.21 ejemplos de aplicaciones famosas que usan barra de navegación). La barra de navegación nos permite en un solo toque cambiar de información, que además su estado de navegación se queda guardado, en contra de los dos que requería en su versión anterior. Además esta disposición contaba con una página principal poco útil ya que simplemente se nos mostraba la portada que hemos hablado antes y un botón de menú. Y para los anuncios es mucho más natural y menos molesto estar colocados en la página principal y no como continuaciones del menú, pudiéndose confundir con este. Como resultado se ha simplificado la navegación y se ha vuelto más intuitiva.
3. Diseño de página: Cada página tiene un *header* o barra superior que indica en que página se encuentra el usuario y si es una página a la que se ha accedido por una de las pilas de navegación contiene un botón de atrás para volver a la anterior página. Las páginas basadas en listas tienen cabeceras del color principal de la aplicación y si tienen una acción a realizar el botón será del mismo color. La información detallada cuenta con una imagen principal que se encoge al hacer *scroll* hacia abajo.

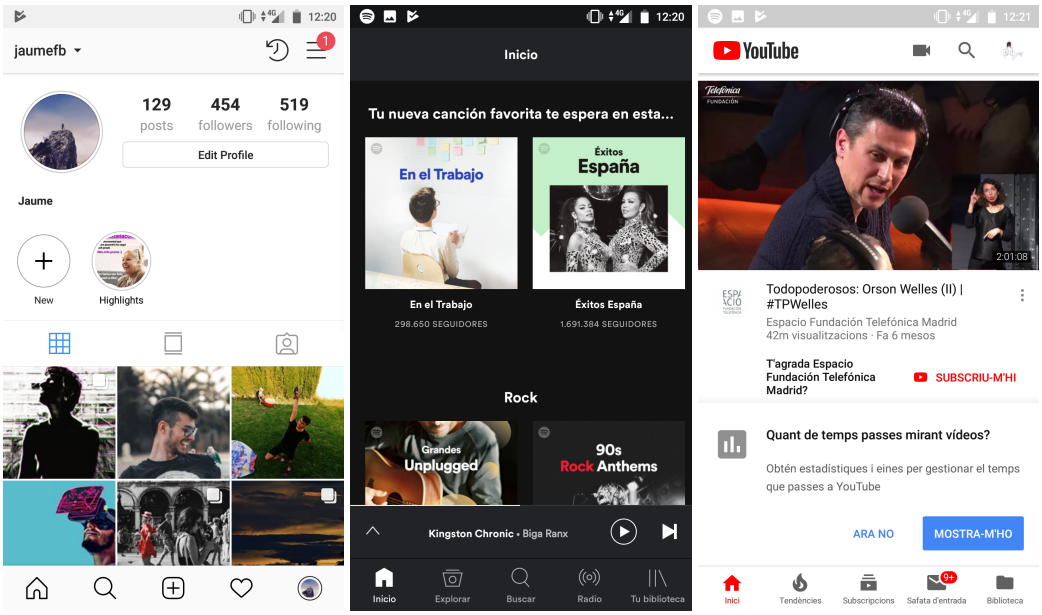

Figura 4.22: Aplicaciones famosas: Instagram, Spotify y Youtube
Todas cuentan con una barra de navegación

CAPÍTULO 5
Ejecución de los Casos de Uso

En este capítulo relacionaremos los casos de uso propuestos en la fase de Análisis en la aplicación terminada, concretamente un escenario para cada tipo de Usuario y cada uno utilizando una plataforma distinta, para poder apreciar los cambios en éstas. Se comentará la navegación y estará acompañada de la explicación de cada página.

5.1 Llamar a un contacto, versión Android

5.1.1. Inicio

Figura 5.1: Pantalla principal de Callosa en Festes

Tras acceder a la aplicación se muestra la pantalla de inicio, que cuenta con información diversa, sin embargo para nuestro propósito no nos es necesaria, así que seleccionamos el icono 'Contactes' de la barra de navegación inferior y procedemos a navegar a la página de los Contactos.

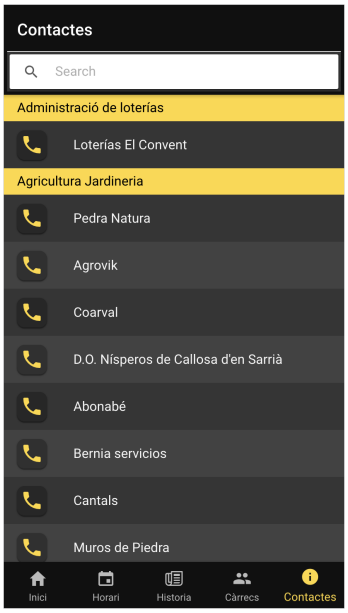

Figura 5.2: Pantalla de contactos de Callosa en Festes

5.1.2. Contactos

Entramos en la lista de los contactos relevantes del pueblo y sus alrededores, siendo esta nuestra página que andábamos buscando. Tras echar un vistazo rápido no encontramos el contacto que precisamos, así que nos dirigimos a la barra superior dónde se aprecia una barra buscadora. Empezamos a escribir en ella. El listado de todos los contactos corresponde al caso de uso CU 08.

5.1.3. Buscar un contacto

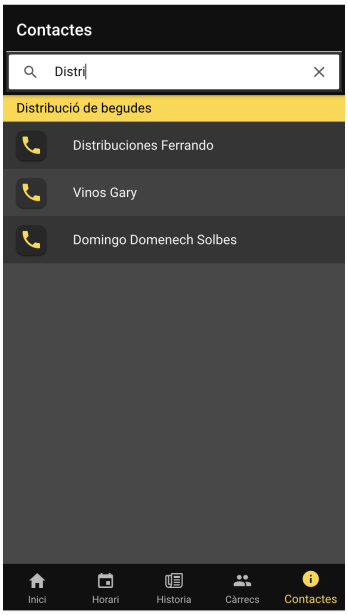

Figura 5.3: Pantalla de Contactos con la lista filtrada

Tras terminar de escribir la palabra que corresponde con el contacto o el tipo de contacto que necesitamos se no muestra una sublista con todos los términos que coinciden

con el patrón de búsqueda. La búsqueda de un contacto corresponde a el caso de uso CU 04.

5.1.4. Llamar a un Contacto

Figura 5.4: Pantalla de llamar a un contacto externa a la app

Al terminar la búsqueda y presionar el icono de el teléfono la aplicación hará uso de el dispositivo para realizar una llamada como si hubieras marcado el número. El llamar a un contacto corresponde a el caso de uso CU 09.

5.2 Editar un cargo, Versión iOS

5.2.1. Inicio

Figura 5.5: Pantalla principal de Callosa en Festes

Tras acceder a la aplicación se muestra la pantalla de inicio, que cuenta con información diversa, entre ella una foto de bienvenida, patrocinadores de la aplicación y finalmente arriba a la derecha un botón que pone logIn, que será nuestro objetivo.

5.2.2. Acreditarse

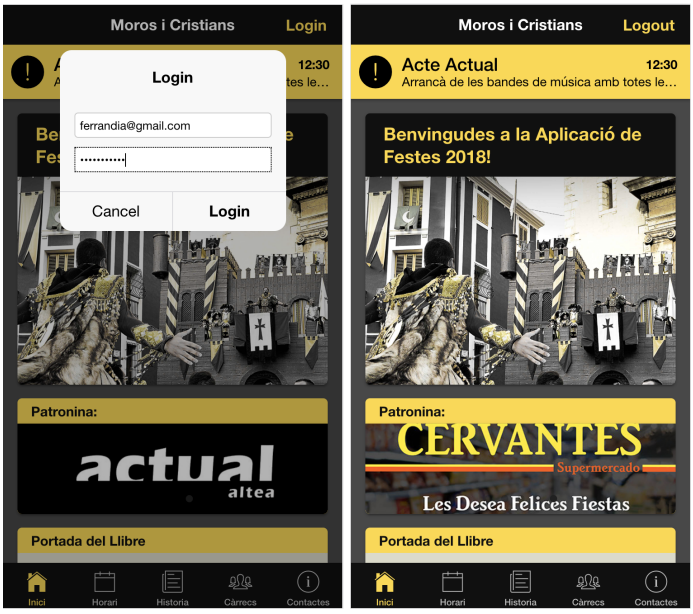

Figura 5.6: Ventana emergente de Acreditación y Pantalla Principal

Tras presionar el botón logIn una ventana emergente es mostrada en la que se nos pide introducir un usuarios y una contraseña, correspondientes a nuestro usuario. Tras intro-

ducirlas y esperar a la respuesta afirmativa de validación del usuario se nos devuelve a la página principal de nuevo, pero ahora el botón de logIn se ha convertido en logOut. Presionamos el botón de 'Càrrecs' para navegar a la siguiente pantalla. La acreditación corresponde al caso de uso CU 11

5.2.3. Listar cargos

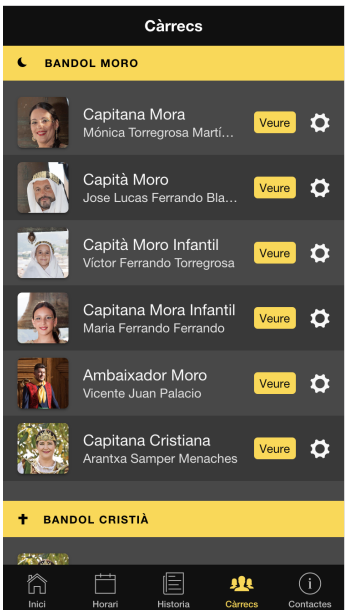

Figura 5.7: Pantalla de cargos de Callosa en Festes

Entramos en la pantalla donde se listan todos los cargos relevantes de las Fiestas de Moros i Cristianos, pero si nos fijamos una cargo, en concreto la capitana cristiana está en el bando equivocado. Como somos administradores accionamos el botón con un engranaje. El listado de los cargos corresponde a el caso de uso CU 05.

5.2.4. Editar un cargo

Una ventana emergente con los datos actuales de el cargo seleccionada, en este caso la capitana cristiana, aparece en pantalla. Localizado el campo equivocado se procede a cambiarlo y seguidamente presionamos el botón 'Acceptar'. Los datos quedan guardados en la base de datos y ya son persistentes para todos los usuarios. La edición de un cargo corresponde al caso de uso CU 13.

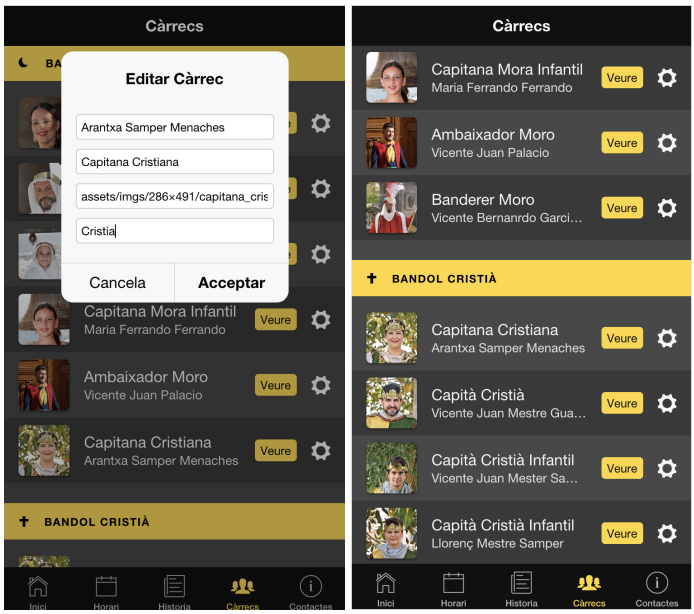

Figura 5.8: Ventana emergente de edición de cargo y pantalla de cargos

CAPÍTULO 6
Conclusiones

El desarrollo de una aplicación desde cero ha supuesto un gran reto, ya tenia poca experiencia en ese ámbito, sumado al tener que de aprender muchas tecnologías nuevas y la escasez de tiempo disponible, ya que había que compaginar el trabajo con el proyecto; sin embargo ha sido cuento menos, gratificante. La curva de aprendizaje y de eficiencia ha sido logarítmica ya que al inicio del proyecto una funcionalidad costaba mucho tiempo y en la recta final del proyecto todo lo contrario, además de que la utilización de buenas prácticas hizo que cualquier cambio resultara sencillo de implementar. Por ejemplo, al tener variables globales para la ruta al backend o para los colores, si se deseaba alterar uno de estos solo se precisaba de cambiar el contenido de una variable.

Durante el camino ha habido muchas experiencias frustrantes, sobre todo en el ámbido fuera de mi zona de confort, la parte de servidor. Sin embargo con paciencia, búsquedas largas por la red y alguna consulta a algún amigo resolvieron con éxito todas estas. Además, el crear un nuevo proyecto ha permitido la implentación de tecnologías a la orden del día y su aprendizaje, abriendo así nuevas puertas para mi futuro laboral.

En la relación del proyecto con la carrera cursada se puede decir que la mayoría de las tecnologías utilizadas han sido aprendidas fuera de el entorno de ésta. Los conceptos generales, la arquitectura y algunos lenguajes si que provienen de cursar el Grado, como el lenguaje SQL, el lenguaje Java y unas pequeñas nociones de JavaScript. El conocimiento de Java ha ayudado a comprender otros lenguajes, en este caso el paso de JavaScript, que no utiliza tipos ni clases, a TypeScript, que sí que los utiliza. Por otra parte todo lo relacionado con el frontend, el html, el CSS, la comunicación entre el frontend y el backend, el diseño de una interfaz móvil han sido aprendidas en el trabajo, o por proyectos personales. Algunos conceptos de crear una aplicación móvil se conocieron en una asignatura llamada Proyecto de Software, pero siendo ésta de tecnología a escoger y en grupos de 8 personas, eligiendo por mayor conocimiento general usar Android Studio para construir una app de una red social. La puesta en marcha de un servidor se dió en una asignatura de la rama de Ingeniería del Software, aunque usando la tecnología Kitematic se ahorró trabajo, y el patrón de modelo, vista, controlador aplicado al backend se usó muy parecido en Java en la asignatura Ingeniería del Software. Algunos conceptos de diseño también se estudiaron en Interfaz Persona Computador, sin embargo la aplicación al móvil de estos y de nuevos se obtenieron en un proyecto de una aplicación móvil externa a la carrera.

En resumen, estoy muy satisfecho con el trabajo realizado, y además el hecho de que se vaya a publicar la aplicación final en sus respectivas plataformas y con el agradecimiento de la Asociació de Moros i Cristians de mi pueblo y su visto bueno hace que en este proyecto de fin de grado el tiempo invertido servirá tanto para mi aprendizaje personal como para mejorar la experiencia de nuestras Fiestas locales.

6.1 Trabajos Futuros

La aplicación presentada para este TFG es una versión estable de lo que será su versión final para este año, ya que la fecha de entrega no coincide con la fecha de entrega real del trabajo. Este hecho se hace notorio sobre todo en los datos, que están desactualizados, ya que las fotos y cierta información aun no está lista, pero en poco tiempo estará ya disponible. Así que como trabajos futuros, se podría dividir en dos partes: las que se implementarán en el siguiente mes y las que van más allá de la versión actual, ya pensadas para el año posterior.

6.1.1. Versión Final

- Actualizar todos los datos desfasados: Se precisa de actualizar el horario y los cargos a los actuales, cosa que ahora resulta mas sencilla mediante el usuario administrador y contando con una base de datos.

- Migrar el servidor: toda la parte del servidor está en el dominio local, ya que para el desarrollo pareció más sencillo, pero para que tenga validez ha de migrarse, ya sea a un servidor contratado o un servicio como AWS de Amazon. Gracias a la tecnología FlyWay la migración será muy sencilla, ya que contamos con todos las versiones de la Base de Datos desde su creación y solo hay que dejar las ejecute secuencialmente.

- Asignar a un Administrador de la Junta: Añadir en el servicio Okta a un miembro de la Junta para que se encargue de cualquier actualización futura de los datos.

- Guardar la publidad en la Base de Datos: Cuando esté claro que publicidad se usará en la aplicación, se editaran las imagenes y se almacenarán en la base de datos.

6.1.2. Versión Futura

- Añadir sección de Kábilas: recopilar información de las diferentes kábilas y crear una página informativa de estas. Esta sección contará ademas una parte editable por un encargado de cada kábila que podrá editar a su gusto, o el que se decida en una junta de ésta. Esta funcionalidad nade de la caracterísitca más de mandada por los festers: un sitio donde poder acceder a los menús que se servirán durante los cuatro días de las fiestas en cada kábila.

- Crear un usuario por cada kábila: Para poder realizar la anterior característica se precisará de la creación de un administrador de cada sección de kábila. A este se le pasará su usuario que cambiará cada año con al renovación anual de los cargos de cada una.

- Permitir acceder con usuario de Facebook: Esta característica esta implementada, pero no se hace uso de ella. la idea es que mediante la autenticación por Facebook los usuarios puedan comentar diferentes secciones o hacer peticiones. Estos usuarios no podrán alterar ningún contenido.

- Sincronizar la app con la página web: El objetivo de esta mejora sería unificar las dos partes para que tanto en diseño como lugar de donde obtiene la inforamción, las Bases de datos, sean iguales. Así cambiando desde un lugar centralizado los datos se actualizarían en todo el set de productos.

- Añadir una opción de visualización de el libro de fiestas: permitir acceder a todas los libros de fiestas de Callosa d'en Sarrià, y principalmente al del año en vigor.
- Crear un lugar donde se puedan subir fotos de fiestas: Crear un espacio donde se selecciones anualmente las mejores fotos de el año de fiestas.
- Implantar un sección de noticias: Añadir un banner de noticias para mantener informados a los usuarios y ver los cambios nuevos de cada año.

Bibliografía

[1] Piattini Mario G. *Análisis y diseño detallado de aplicaciones informáticas de gestión*. RAMA Editorial, Madrid, primera edición, 1996.

[2] Especificación de Requisitos Software según el estándar de IEEE 830 Consultado el 2/08/18 http://zeus.inf.ucv.cl/~bcrawford/AULA_ICI_3242/ERS_IEEE830.pdf.

[3] La importancia del mock-up en los proyectos de IT Consultado el 10/08/18 https://www.workana.com/blog/uncategorized/importancia-mock-up-proyectos-it/.

[4] Flyway (software) Consultado el 11/08/18 https://en.wikipedia.org/wiki/Flyway_(software).

[5] What is the structure of a Mobile App? Consultado el 12/08/18 https://www.quora.com/What-is-the-structure-of-a-Mobile-App.

[6] ¿Qué es Spring Framework? Consultado el 15/08/18 https://www.genbeta.com/desarrollo/que-es-spring-framework.

[7] Web App Architecture - the Spring MVC - AngularJs stack Consultado el 15/08/18 https://blog.angular-university.io/developing-a-modern-java-8-web-app-with-spring-mvc-and-angularjs/.

[8] Las 10 principales diferencias entre AngularJS y Angular Consultado el 15/08/18 https://www.campusmvp.es/recursos/post/las-10-principales-diferencias-entre-angularjs-y-angular.aspx.

[9] The Importance of Good Software Architecture Consultado el 15/08/18 https://dzone.com/articles/the-importance-of-a-good-software-architecture.

[10] Ionic 2, un framework diseñado para las aplicaciones híbridas que da el salto a las aplicaciones web Consultado el 16/08/18 https://www.arsys.es/blog/programacion/ionic-2/.

[11] The color system Consultado el 20/08/18 https://material.io/design/color/the-color-system.html#.

[12] IntelliJ IDEA Consultado el 21/08/18 https://es.wikipedia.org/wiki/IntelliJ_IDEA.

[13] VERSUS: IntelliJ IDEA VS Eclipse Consultado el 21/08/18 https://www.paradigmadigital.com/dev/versus-intellij-idea-vs-eclipse/.

[14] VERSUS: Back-End Web Architecture Consultado el 24/08/18 https://www.codecademy.com/articles/back-end-architecture.

[15] 9 Steps: Choosing a tech stack for your web application Consultado el 30/08/18 https://medium.com/unicorn-supplies/9-steps-how-to-choose-a-technology-stack-for-your-web-application-a6e302398e55.

[16] How to Choose a Technology Stack for Web Application Development Consultado el 30/08/18 https://rubygarage.org/blog/technology-stack-for-web-development.

[17] ¿Qué importancia tiene el diseño en el desarrollo de una App? Consultado el 01/09/18 https://slashmobility.com/blog/2014/04/que-importancia-tiene-el-diseno-en-el-desarrollo-de-una-app/.

[18] Framework Consultado el 8/09/18 https://es.wikipedia.org/wiki/Framework.

APÉNDICE A
Elementos del Frontend

En este apéndice se muestra un ejemplo de cada elemento del frontend y su código.

A.1 Page

como ejemplo usaremos la Contactes Page.

- contactes.html

```
<ion-header>

  <ion-navbar>
    <ion-title class="title-contactes">Contactes</ion-title>
    <ion-searchbar (ionInput)="filterContactes($event)" class="searchbar-contactes"></ion-searchbar>
  </ion-navbar>

</ion-header>


<ion-content>
    <ion-list>
      <ion-item *ngFor="let item of contactes; let i = index " [attr.data-index]="i" [ngClass]="{'card-pair': i%2 == 0 && item.header == undefined,
       'card-not-pair': i%2 != 0 && item.header == undefined, 'header-lista': item.header == 'yes'}" class="contacte"
      (click)="item.header == undefined && presentAlert(item.nom, item.telefono)">
        <h2 class="contacte-name"> {{item.nom}}</h2>
        <ion-icon name='call' item-start *ngIf="item.header==undefined && item.nom!='' " [ngClass]="{'tlf-icon-ios': plt.is('ios'),
       'tlf-icon-md': plt.is('android')}"
                  (click)="call(item.telefono)"></ion-icon>
        <ion-icon (click)="presentPrompt(item)" name="settings" item-end color="black" *ngIf="item.header == undefined && oauthService.hasValidIdToken()"></ion-icon>
      </ion-item>
    </ion-list>
</ion-content>
```

- contactes.ts

```
import { Component } from '@angular/core';
import {AlertController, IonicPage, NavController, NavParams, Platform}
    from 'ionic-angular';
```

```
import {ContactesDataProvider} from "../../providers/contactes-data/contactes-data";
import {CallNumber} from "@ionic-native/call-number";
import {OAuthService} from "angular-oauth2-oidc";


/**
 * Generated class for the ContactesPage page.
 *
 * See https://ionicframework.com/docs/components/#navigation for more info on
 * Ionic pages and navigation.
 */

@IonicPage()
@Component({
  selector: 'page-contactes',
  templateUrl: 'contactes.html',
})
export class ContactesPage {

  contactes: any;

  isSearching = false;

  contactesBase: any;

  constructor(public navCtrl: NavController, public navParams: NavParams, public contactesData: ContactesDataProvider,
              private callNumber: CallNumber, public plt: Platform, private alertCtrl: AlertController,
              private oauthService: OAuthService) {

    console.log(this.contactes)
  }



  ngOnInit(){
    this.contactesData.getContactes().subscribe(contactes => {
      this.contactes = contactes.sort(function (a, b)    {
        a = a.tipo.toLowerCase();
        b = b.tipo.toLowerCase();
        if (a > b) return 1;
        if (a < b) return -1;
        return 0;
      });
      var header = "a";
      this.contactes.forEach((el, index) =>{
        if(!(el.tipo == header)){
          header = el.tipo
          this.contactes.splice(index, 0, {nom:el.tipo,header:"yes",tipo:el.tipo});


        }

      })
      this.contactesBase = this.contactes;
    })
  }



```

```
62  ionViewDidLoad() {
63    console.log('ionViewDidLoad ContactesPage');
64  }
65
66  call(num){
67    this.callNumber.callNumber(num, true)
68  }
69
70  filterContactes(ev:any){
71    this.contactes = this.contactesBase
72    const val = ev.target.value;
73    if(val == "")
74      this.isSearching = false
75    else
76      this.isSearching = true
77    if (val && val.trim() != '') {
78      this.contactes = this.contactes.filter((item:any) => {
79        console.log(item.tipo)
80        return ((item.nom.toLowerCase()
81          .indexOf(val.toLowerCase()) > -1) || (item.tipo.toLowerCase()
82          .indexOf(val.toLowerCase()) > -1));
83      })
84    }
85
86  }
87
88  presentAlert(nom, telefono) {
89    event.stopPropagation()
90    let alert = this.alertCtrl.create({
91      title: nom,
92      message: telefono,
93      buttons: ['OK']
94    });
95    alert.present();
96  }
97
98  presentPrompt(item) {
99    event.stopPropagation()
100   let self = this
101   let alert = this.alertCtrl.create({
102     title: 'Editar Carrec',
103     inputs: [
104       {
105         name: 'nom',
106         placeholder: 'Nom',
107         value: item.nom
108       },
109       {
110         name: 'telefono',
111         placeholder: 'Telefono',
112         value: item.telefono
113       },
114       {
115         name: 'tipo',
116         placeholder: 'Tipo',
117         value: item.tipo
118       }
119
120     ],
121     buttons: [
122       {
123         text: 'Cancela',
124         role: 'cancela',
125         handler: data => {
```

```
126
127             }
128           },
129           {
130             text: 'Acceptar',
131             handler: data => {
132               data.id = item.id
133               this.contactesData.postContacte(data)
134                 let index = this.contactesBase.findIndex(function(contacte)
                      {return contacte.id == data.id})
135               this.contactesBase.splice(index,1,data)
136               this.contactes = this.contactesBase
137
138             }
139           }
140         ]
141       });
142       alert.present();
143     }
144
145 }
```

- contactes.scss

```
1  page-contactes {
2    .content{
3      background-color: #484848;
4    }
5
6    .card-pair{
7      background-color: #424242;
8      color: #f8f8f8;
9    }
10   .card-not-pair{
11     background-color: #353535;
12     color: #f8f8f8;
13   }
14
15   .list-md .item-block .item-inner{
16     border: 0px;
17   }
18   .list-ios .item-block .item-inner{
19     border: 0px;
20   }
21
22   .header-lista{
23     background-color: background-color: color($colors,primary);
24     color: #000000;
25     min-height: 1px;
26     height: 2.5em;
27     font-size: 13px;
28     font-weight: bold;
29   }
30
31   .tlf-icon-ios{
32     background-color: #00000045;
33     border-radius: 9px;
34     padding: 2px;
35     width: 33px;
36     color: background-color: color($colors,primary);
37     box-shadow: 0px 1px 1px #000000a3;
38   }
39
40   .tlf-icon-md{
```

```
41    background-color: #00000045;
42    border-radius: 9px;
43    padding: 4px;
44    width: 36px;
45    color: background-color: color($colors,primary);
46   -webkit-box-shadow: 0px 1px 1px #000000a3;
47    box-shadow: 0px 1px 1px #000000a3;
48  }
49
50    .contacte-name{
51      white-space: nowrap;
52      overflow: hidden;
53      text-overflow: ellipsis;
54    }
55
56
57
58
59    .content .scroll-content{
60      margin-top: 56px;
61
62    }
63
64
65    .searchbar-contactes{
66      border-top: 1px solid #ffffffd4;
67      width: 100%;
68      background-color: #353535;
69
70    }
71
72    .title-contactes{
73      height: 52px;
74      position: relative;
75    }
76
77    .toolbar .searchbar-ios .searchbar-input{
78      background-color: white;
79    }
80
81  }
```

A.2 Provider

- contactes-data.ts:

```
1  import { HttpClient } from '@angular/common/http';
2  import { Injectable } from '@angular/core';
3  import { Observable} from "rxjs/Observable";
4  import {RequestOptions} from "@angular/http";
5  import {ConstantsProvider} from "../constants/constants";
6  import { RUTA_LOCAL} from "../constants/constants";
7
8  /*
9    Generated class for the ContactesDataProvider provider.
10
11   See https://angular.io/guide/dependency-injection for more info on
         providers
12   and Angular DI.
13 */
14 @Injectable()
15 export class ContactesDataProvider {
```

```
16
17    public API = "";
18
19
20    constructor(public http: HttpClient, public constants:
          ConstantsProvider) {
21      this.API = RUTA_LOCAL
22
23
24
25    }
26
27    result:any;
28
29      getContactes():Observable<any> {
30        return this.http.get(this.API + '/contactes')
31      }
32
33      postContacte(contacte){
34
35
36        this.http.post(this.API + '/contactes',contacte).subscribe(
37          res => {
38            console.log(res);
39          },
40          err => {
41            console.log("Error occured");
42          }
43        );
44      }
45
46
47  }
```

APÉNDICE B
Elementos del backend

En este apédice se muestra una ejemplo de cada elemento del backend y su código. Se usará como modelo el caso de un contacto

- DAO:

```
1  package com.callosa.mic.cad.daos;
2
3  import com.callosa.mic.cad.interfaces.IContacteRepository;
4  import com.callosa.mic.model.Contacte;
5  import org.springframework.beans.factory.annotation.Autowired;
6  import org.springframework.context.annotation.Configuration;
7
8  import java.util.ArrayList;
9  import java.util.List;
10 import java.util.Optional;
11
12 @Configuration
13 public class ContacteDAO {
14
15     @Autowired
16     private IContacteRepository contacteRepository;
17     //Devuelve todos los contactos
18     public List<Contacte> getContactes(){
19
20         List<Contacte> contactes = new ArrayList<Contacte>();
21
22         try{
23
24             contactes = (List<Contacte>) contacteRepository.findAll();
25
26         } catch (Exception e){
27             System.out.println("ContacteDAO-getContactes():"+e.getMessage
                   () + '-'+e.getCause());
28             e.printStackTrace();
29         }
30
31         return contactes;
32     }
33
34     public Contacte setContacte(Contacte contacte_nou)
35     {
36         Contacte contacte = null;
37
38         try{
39
40             contacte = contacteRepository.save(contacte_nou); //Save or
                   update
```

```
41
42          }catch (Exception e){
43
44              System.out.println("ContacteDAO-setContacte():"+e.getMessage
                    () + '-'+e.getCause());
45              e.printStackTrace();
46          }
47
48          return contacte;
49
50      }
51
52      public Contacte getContacteById(Long idContacte){
53
54          Contacte contacte = null;
55
56          try{
57
58              Optional<Contacte> optionalContacte = contacteRepository.
                    findById(idContacte);
59              contacte = optionalContacte.get();
60
61          }catch (Exception e){
62
63              System.out.println("ContacteDAO-getContacteById():"+e.
                    getMessage() + '-'+e.getCause());
64              e.printStackTrace();
65          }
66
67          return contacte;
68      }
69  }
```

- Interface:

```
1  package com.callosa.mic.cad.interfaces;
2
3  import com.callosa.mic.model.Contacte;
4  import org.springframework.data.repository.CrudRepository;
5
6  public interface IContacteRepository extends CrudRepository<Contacte,
       Long> {
7
8  }
```

- Model:

```
1  package com.callosa.mic.model;
2
3  import org.apache.commons.lang3.builder.EqualsBuilder;
4  import org.apache.commons.lang3.builder.HashCodeBuilder;
5  import org.apache.commons.lang3.builder.ToStringBuilder;
6
7  import javax.persistence.Entity;
8  import javax.persistence.GeneratedValue;
9  import javax.persistence.Id;
10 import javax.persistence.Table;
11
12 @Entity
13 @Table(name = "CONTACTES")
14 public class Contacte {
15
16     @Id
17     @GeneratedValue
```

```
18      private Long idContacte;
19
20      private String nom;
21
22      private String descripcio;
23
24      private String telefono;
25
26      public Contacte(){}
27
28      public Contacte(String nom, String descripcio, String telefono) {
29          this.nom = nom;
30          this.descripcio = descripcio;
31          this.telefono = telefono;
32      }
33
34      public Long getIdContacte() {
35          return idContacte;
36      }
37
38      public void setIdContacte(Long idContacte) {
39          this.idContacte = idContacte;
40      }
41
42      public String getNom() {
43          return nom;
44      }
45
46      public void setNom(String nom) {
47          this.nom = nom;
48      }
49
50      public String getDescripcio() {
51          return descripcio;
52      }
53
54      public void setDescripcio(String descripcio) {
55          this.descripcio = descripcio;
56      }
57
58      public String getTelefono() {
59          return telefono;
60      }
61
62      public void setTelefono(String telefono) {
63          this.telefono = telefono;
64      }
65
66      @Override
67      public boolean equals(Object o) {
68          if (this == o) return true;
69
70          if (!(o instanceof Contacte)) return false;
71
72          Contacte contacte = (Contacte) o;
73
74          return new EqualsBuilder()
75                  .append(idContacte, contacte.idContacte)
76                  .append(nom, contacte.nom)
77                  .append(descripcio, contacte.descripcio)
78                  .append(telefono, contacte.telefono)
79                  .isEquals();
80      }
81
```

```
    @Override
    public int hashCode() {
        return new HashCodeBuilder(17, 37)
                .append(idContacte)
                .append(nom)
                .append(descripcio)
                .append(telefono)
                .toHashCode();
    }

    @Override
    public String toString() {
        return new ToStringBuilder(this)
                .append("idContacte", idContacte)
                .append("nom", nom)
                .append("descripcio", descripcio)
                .append("telefono", telefono)
                .toString();
    }
}
```

- Controller:

```
package com.callosa.mic.rest;

import com.callosa.mic.cad.daos.ContacteDAO;
import com.callosa.mic.model.Contacte;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@CrossOrigin
@RestController
public class ContacteController {

    @Autowired
    ContacteDAO contactesDAO;


    /**
     * Devuelve los contactos de bd
     * @return
     * lista de contactes
     */

    @RequestMapping(value="/contactes",method= RequestMethod.GET)
    public List<Contacte> fetchContactes(){

        return contactesDAO.getContactes();
    }

    /**
     * Devuelve el contacto por Id
     * @return
     * contacte
     */

    @RequestMapping(value="/contactes/{idContacte}",method= RequestMethod
        .GET)
    public Contacte getContacte(@PathVariable Long idContacte){

        return contactesDAO.getContacteById(idContacte);
    }
```

```
41
42      @RequestMapping(value="/contactes",method= RequestMethod.POST)
43      public Contacte addCliente(@RequestBody Contacte contacte){
44
45          return contactesDAO.setContacte(contacte);
46      }
47  }
```

- Flyway Version:

```
1  CREATE TABLE  contactes  (
2     id_contacte  bigint(20) NOT NULL AUTO_INCREMENT,
3     nom  varchar(400) NOT NULL,
4     descripcio  varchar(2600) DEFAULT NULL,
5     telefono  varchar(20) DEFAULT NULL,
6    PRIMARY KEY ( id_contacte )
7    );
```